Biotic Prediction

Building the Computational Technology Infrastructure for Public Health and Environmental Forecasting

Software Design Document

BP-SDD-1.5

Task Agreement: GSFC-CT-1

February 20, 2004

1 INTRODUCTION ... 1
1.1 INVASIVE SPECIES FORECASTING SYSTEM ... 1
1.2 SOFTWARE DESIGN DOCUMENT OVERVIEW ... 1
1.3 DOCUMENT VERSIONS ... 1
1.4 REFERENCED DOCUMENTS ... 1
2 DESIGN OVERVIEW ... 3
2.1 DESIGN DRIVERS ... 3
2.2 SYSTEM DESIGN ... 3
2.3 EXTERNAL INTERFACES ... 3
2.4 CONCEPTUAL DESIGN ... 3
2.5 REUSE STRATEGY ... 4
2.6 SECURITY STRATEGY ... 5
2.6.1 User Profiles ... 5
2.6.2 Communications Security ... 5
2.6.3 Data Security ... 5
2.7 RISK ASSESSMENT ... 5
2.8 PERFORMANCE CONSIDERATIONS ... 6
2.8.1 Approach & Benchmarks ... 6
2.8.2 Platform Specs ... 6
2.9 DEVELOPMENT ENVIRONMENT ... 7
3 SYSTEMS DESCRIPTION ... 8
3.1 ISFS SUBSYSTEMS ... 8
3.1.1 Front End Layer ... 8
3.1.2 Application Layer ... 8
3.1.3 Back End Layer ... 11
3.2 USER ROLES ... 12
3.3 INPUT/OUTPUT ... 12
3.3.1 Input Files ... 12
3.3.2 Output Files ... 13
3.4 SYSTEM OPERATIONS ... 13
3.4.1 Restrictions ... 13
3.4.2 Compute Server ... 13
3.4.3 Operational Scenario ... 14
3.4.4 Model Run Use Case ... 15
4 OBJECT-ORIENTED DESIGN ... 17
4.1 INTRODUCTION ... 17
4.2 PACKAGING ... 17
4.3 MODEL RUN DESIGN ... 18
4.3.1 Overview and Responsibilities ... 18
4.3.2 Class Descriptions ... 19
4.3.3 Model Run Class Diagrams ... 19
4.4 USER ACCOUNTS DESIGN ... 24
4.4.1 Overview and Responsibilites ... 24
4.4.2 Class Descriptions ... 24
4.4.3 User Account Class Diagrams ... 24
4.5 ISFS COMMON DESIGN ... 26
4.5.1 Overview and Responsibilities ... 26
4.5.2 Class Descriptions ... 26
4.5.3 Utility Diagrams ... 26
4.6 HIERARCHY FOR ALL PACKAGES ... 28
4.7 CLASS HIERARCHY ... 29

5 DATA DICTIONARY 30

5.1 RELATIONAL DATA TABLES 30

5.1.1 krigparams 30

5.1.2 krigroutine 30

5.1.3 mergeddataset 31

5.1.4 modelarray 31

5.1.5 modeloutput 32

5.1.6 modelrun 32

5.1.7 studysite 32

5.1.8 useraccount 33

5.2 ENTITY RELATIONSHIP (ER) DIAGRAM 33

GLOSSARY 34

APPENDIX A INPUT/OUTPUT SAMPLES 35

MAP OUTPUT 35

LOGGED OUTPUT FROM STEPWISE REGRESSION WITH KRIGING 36

MERGED DATASET SAMPLE 41

MODEL RUN PARAMS XML 42

APPENDIX B DEVELOPER SETUP/DEPLOYMENT INSTRUCTIONS 43

TOOLS AND TECHNOLOGIES OVERVIEW 43

TOOLS AND ENVIRONMENT SETUP 43

CONFIGURING FOR LOCAL DEVELOPMENT 44

RUNNING ISFS 46

DEPLOYMENT 46

APPENDIX C MODEL CODE PERFORMANCE IMPROVEMENT 47

Milestone F — First Code Improvement (Parallel Kriging) 48

Parallel Kriging Results on Ideally Sized Test Cases 49

Evaluation of Community and Advanced Improvement Goals 51

Progress toward Milestones 52

1 Introduction

1.1 Invasive Species Forecasting System

This project is developing the high-performance, computational technology infrastructure needed to analyze the past, present, and future geospatial distributions of living components of Earth environments. This involves moving a suite of key predictive, geostatistical biological models into a scalable, cost-effective cluster computing framework; collecting and integrating diverse Earth observational datasets for input into these models; and deploying this functionality as a Web-based service. The resulting infrastructure will be used in the ecological analysis and prediction of exotic species invasions. This new capability will be deployed at the USGS Fort Collins Science Center and extended to other scientific communities through the USGS National Biological Information Infrastructure program.

1.2 Software Design Document Overview

This Software Design Document has been prepared in accordance with NASA/GSFC's "Recommended Approach to Software Development Revision 3." The core document is organized into five major sections; Project Introduction, Design Overview, Systems Description, Detail Engineering Design and Data Interfaces.

The Design Overview highlights the high-level design and drivers for the design choices. In addition, strategies for reuse, security, performance & risk mitigation are also presented.

The Systems Description discusses the major system components, their processes, operation and data inputs/outputs.

The detailed engineering design may be found in Object-Oriented Design section. Described are web application’s classes, interfaces to the data & compute engine and program run sequences.

A listing of the database tables and their relationships are shown in the Data Dictionary section

Appendices include a Glossary of Terms, developer setup/deployment instructions, model input/output examples and Kriging performance results.

1.3 Document Versions

Date	Version	Description
August 22, 2003	1.1	Initial submission to CT relating to Milestone F
Dec 3, 2003	1.2	Second submission to CT relating to Milestone F:
Jan 26, 2004	1.3	Added Appendix C, Kriging performance improvements for Milestone F
Feb 8, 2004	1.4	Adjusted some figures in Appendix C.
Feb 20, 2004	1.5	Adjusted CFGS figure in Milestone table of Appendix C

1.4 Referenced Documents

Document Title	Version	Date
Software Test Plan (BP-TP)	1.2	2003-12-03
Concept of Operations (BP-CONOP)	1.9	2002-12-04
Software Requirements (BP-SRD)	1.6	2003-11-30
Software Requirements Trace Matrix (BP-SRTM)	1.0	2003-11-30

Baseline Software Design (BP-BSD)	1.3	2002-11-25

2 Design Overview

2.1 Design Drivers

The Invasive Species Forecasting System is one of NASA's Computational Technologies (CT) projects. The CT project mission & objectives (http://ct.gsfc.nasa.gov/overview.html) are as follows:

> “The goal of NASA's Computational Technologies (CT) Project is to demonstrate the potential afforded by teraFLOPS (trillion floating-point operations per second) performance to further our understanding and ability to predict the dynamic interaction of physical, chemical, and biological processes affecting the solar-terrestrial environment and the universe.”

The ISFS project is driven by five specific CT objectives:

- Support the development of massively parallel, scalable, multidisciplinary models and data processing algorithms.
- Make available prototype, scalable, parallel architectures and massive data storage systems to CT researchers.
- Prepare the software environments to facilitate scientific exploration and sharing of information and tools.
- Develop data management tools for high-speed access management and visualization of data with teraFLOPS computers.
- Demonstrate the scientific and computational impact for Earth and space science applications.

2.2 System Design

The ISFS system will have users from government agencies, universities, industry, and the public. To the users, the ISFS is deployed as a web browser accessible system that will present options for applying a series of models to available datasets yielding predictive result sets. The system can ingest data from different sources and in different formats and existing models can be either run, or new ones created. The system outputs maps and additional information depicting the applied model and the predicted species distributions.

2.3 External Interfaces

The primary means of interface to the ISFS will be a W3 standards compliant web browser. All GUI development efforts for external interfaces will be through a web-based client architecture that uses HTTP as the primary means for interacting with the system. Supplementary interfaces for ingest may include a secured FTP push/pull technique that will be scheduled through the GUI.

Standing orders for required data products will be accomplished through subscriptions to the varied data sources including the DAACs, NBII, USGS, and other data sources TBD. Different levels of access for public, model user, model builder, and developer will be established and documented in the Software Requirements Document (SRD).

The user interface is implemented through an HTTP connection to a browser-based GUI to select models and datasets, and to apply the selected model to the selected dataset. The GUI is also the vector for returning processed output to the end user.

2.4 Conceptual Design

The overall conceptual design of the Invasive Species Forecasting System is presented in Figure 1. It consists of three major layers:

Front End Layer

The Front End layer will support a variety of interfaces that allow controlled and tailored access to the various subsystems of the overall system. The front end consists of a graphical user interface subsystem with both client- and server-side components.

The Web Interface provides a way of managing information and performing analyses by means of dynamically constructed user workspaces, or role-based views. The ISFS has no client-side applets or controls, and relies on the inherent capabilities of a browser to support client-side Javascript, DHTML & XML.

Application Layer

The Applications layer provides features in support of primary system functions; data ingest and pre-processing, invoking & processing modeling routines & post-processing (data visualization, decision support functions).

Back End Layer

The Back End layer serves 2 main functions; it is a compute engine and an archive system. The compute engine is configured as a Beowulf cluster to provide the high performance computing needed to run the parallel kriging routines. The cluster accesses the archive system, which is a large disk array, providing persistent storage for both system and user needs. The compute server processes the model algorithms, the archival system stores all data used by and generated by the modeling routines & the web application.

Each of these major layers and their subsystems are explained in greater detail in the Systems Descriptions. As well, the Concept of Operations, Internal Architecture section provides workflow, processing steps & detailed description of each application function.

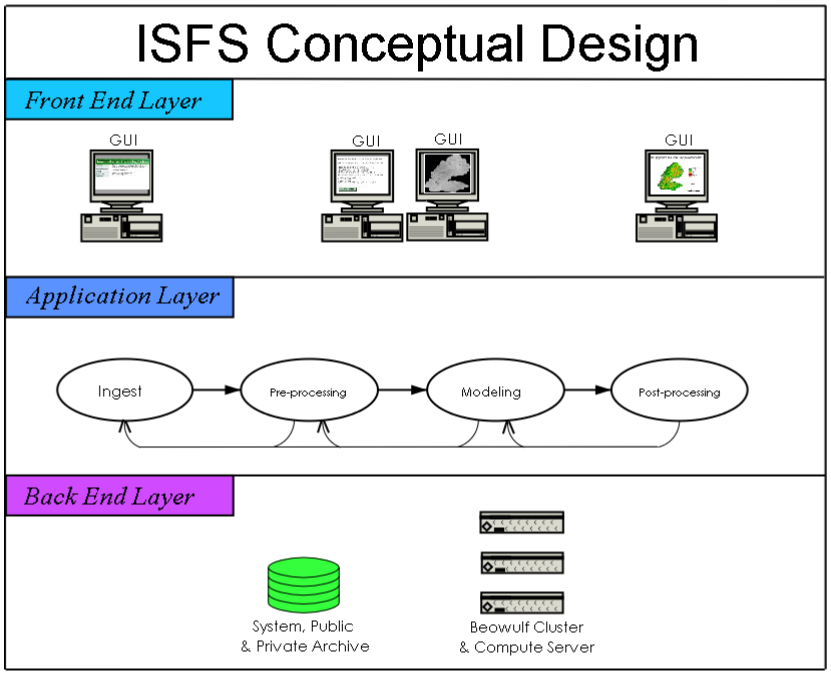


Figure 1 - Conceptual Design

2.5 Reuse Strategy

When designing and implementing ISFS, reuse will be an important goal, and all attempts will be made to create components and develop modular, encapsulated and abstract objects and data structures. We are evaluating integrating Earth Clearing HOuse (ECHO) and NBII APIs into the data ingest portion of the system. The user management logic seems to hold particular promise for design using reusable libraries. Reasonable effort will be made to mimic this design method on other classes when appropriate to result in a solid design with maximized reuse at both the code level and at the component level.

The Invasive Species Forecasting System is taking advantage of reuse in a number of ways. First we are looking to use various software, components and libraries that are directly applicable to building our system quickly and economically. One large area of reuse is with regards to various open source and COTS packages that are incorporated into the ISFS. The use of the Struts library is a good case in point where we can reuse a Model/View/Controller library and extend it into our problem domain. IDL/ENVI, a COTS package, allows for easy data manipulation and contains an extensive library of data management and statistical routines. These built-in libraries and a user contributed IDL library are both being used. This allows us to leverage off of the work already done by others. For a full list of other libraries and packages used, see Appendix B.

The design of the ISFS is not discipline specific, and may be applicable for other statistical modeling purposes. In particular, the data ingest and data display systems are suitable for incorporation into other data analysis systems. Data ingest and analysis are common subsystems in Earth science applications, and successful implementation of these systems in the ISFS will be readily accepted by other similar projects.

2.6 Security Strategy

2.6.1 User Profiles

Each user is assigned a particular role, such as Administrator, Model Builder or Model User, which enables them to perform certain actions and to view certain types of information. Section 3.4.1, System Operations Restrictions describes these authorization levels in detail.

2.6.2 Communications Security

Communications security between the front end and the application server use standard HTTP security. The communications protocol between the application layer and the back end will be RMI. RMI contains its own built-in security model that we will build upon.

There is an additional measure that will be used to improve security and limit uncontrolled access to the cluster. The host node of the cluster should be configured to only accept RMI connections from the specific server where the application is running, thus limiting access by only that machine.

2.6.3 Data Security

In most respects the ISFS is an open system and data on the archive will be visible to all users. Users must register with the system before having access to any of the data within the archives. If a user chooses to keep their data private, the system will support a private archive where the data is only visible to the owner of the data. Depending on user needs, future versions may include data security at a group level. A group of users can access data within the group, but to those users outside the group the data would not be visible. The permissions on the file will be maintained in a data registry within the RDBMS.

2.7 Risk Assessment

Risks in the current design & implementation strategy exist [note: this list is dynamic and will change as the project evolves]:

- A secure & standard protocol for communicating though a firewall to the cluster must be defined. The prototype currently uses SSH which doesn't provide seamless, asynchronous communication between the application and the back end layer.
- Requirements may be stale and not helpful to the current needs of our user community .
- Disparate nature of the scientific material the system is to support, such as dynamic and interchangeable modeling algorithms and the diverse nature of the ingested data, lends itself to a complicated design.

- Design goal to integrate 3rd party technologies such as ECHO and Earth Science Markup Language (ESML) introduces risk in that we're dependent on external parties' delivery schedules, product content and support structures.
- Limited data validation capabilities at this stage
- Modeling algorithms on the compute server have not been tested under concurrent, multi-user sessions.

2.8 Performance Considerations

2.8.1 Approach & Benchmarks

Realizing gains in performance of the ISFS project depends on two primary factors: 1) parallelizing the Modeling-Kriging routines and 2) processing these routines in a cluster computing environment.

The overall performance goal for this code is to improve processing times and increase the amount of data handled by the model. To summarize, the results indicate that we have achieved our Milestone F Community Improvement goal of a 25x speed up on a 32-processor cluster with greater than 75% efficiency. The results also document the general scaling behavior of the kriging algorithm and point to the limits in scalability one might expect as more nodes are allocated to the canonical data sets. The Baseline Design Document [Section 5 - Baseline Scenario & section 6 - Performance Improvement Plan] describes the initial study sties & datasets, details the steps taken to improve the performance of the modeling, specifically the kriging step, and what results have been realized.

2.8.2 Platform Specs

The cluster computing system is configured as a Beowulf-class parallel computer, (a cluster of PCs running LINUX, connected by its own private LAN.)

The clustered computers designated & configured for the science & engineering teams are as follows:

<u>MEDUSA</u> {science development}

128-processor 1.2 GHz Athlon MP cluster
17 coupled developer nodes in GSFC scientist offices
64-gigabyte DDR random access memory
2.56-terabyte disk
2 gigabit-per-second Myrinet internal network
Fast Ethernet network to developer nodes

<u>PIVOT</u> {application engineering}

26 node cluster (16 dual processor, 10 quad processor)
PIII 500Mhz processors w/ 512k cache
Each dual node has 4.5Gb local storage
Each quad node has 9Gb local storage
Myrinet 1Gb/sec internal network

Two identical production systems have been specified & ordered. One will be housed at NASA GSFC and the other at USGS Colorado and are configured as follows:

<u>FIREANT & ROCKYMOUNT</u> {Production Systems}

16 nodes, 32-processor 1.4 GHz AMD Opteron cluster
16-gigabyte SDRAM
1.44 Tb RAIDed disk storage
Gigabit Ethernet internal network

2.9 Development Environment

ISFS is being developed using best practices, COTS and open source tools to manage the development process. The design is being developed using Borland Together to create UML diagrams that result in generated Java code. The application layer is predominantly written using Java. The Back End layer is more heterogeneous, with a mixture of languages such as scripts, IDL/ENVI, C and Fortran. Both the application and model are hosted on Linux platforms. Version control is performed using CVS. Defect tracking is performed using Bugzilla. Testing will play a key role in the development cycle. Junit will be used to create an automated test harness to catch defects as early in the development cycle as possible. Apache and Tomcat are used for the web application. Please refer to Appendix B for a comprehensive listing of the various software tools and description of the development environment.

3 Systems Description

3.1 ISFS Subsystems

The ISFS is organized as illustrated in the figure below. The user interfaces with the web application, presented in the Web browser. The application layer manages the user sessions. When the user is ready to submit a model run, the application makes sure all the data is staged on the back end archive and then executes the model run on the back end Linux cluster.

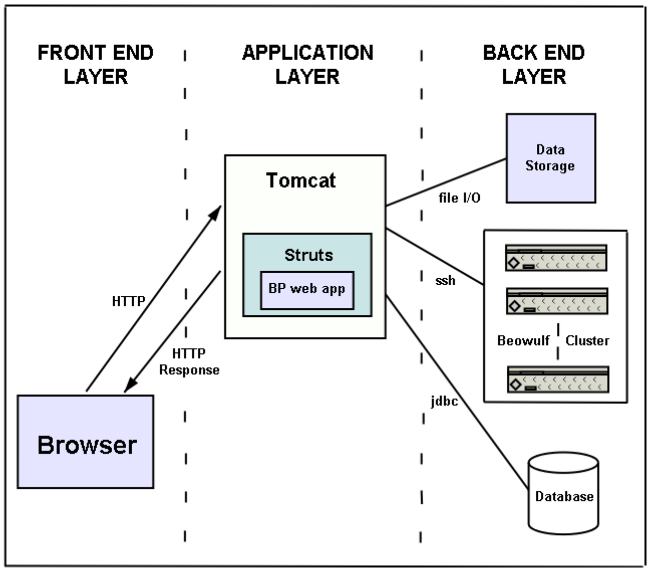


Figure 2 - The browser interacts with the web application to create and configure a model run. The application then stages and runs the model on the back end.

3.1.1 Front End Layer

The front end layer consists of a Web browser interface allowing a user to upload data files into the ISFS, configure a new model run and access model run results. The front end layer is implemented using HTML to allow the widest cross-platform access to the modeling system.

3.1.2 Application Layer

The Application layer gathers user input such as study site and model type from the front end layer via servlets and Java Server Pages (JSP). This information is used to build a complete description of the intended model run. This layer manages the interaction between the user and the model run. Interfaces to the relational database are also managed here. Servlet technology is a primary design strategy for the components of the Application Layer. Jakarta Tomcat is used as the servlet engine and Jakarta Struts as the servlet framework.

Struts is used within Tomcat to implement the model-view-controller pattern and thus help separate the GUI presentation layer in JSP from the business logic found across the Application & Back End layers. Java Beans from the user sessions, user information and other state information are stored in a Relational Database Management System (RDBMS) and accessed using Java Database Connectivity (JDBC). JDBC adds another layer of abstraction from the relational database, as well as providing flexibility to place the RDBMS on either the application server host or the back end host.

As illustrated in the figure below, the application layer consists of four subsystems that support the functionality, computations, and workflows of the ISFS; Ingest, Pre-processing, Modeling, and Post-processing. These subsystems may be invoked by the user sequentially, arbitrarily, and iteratively to support the various steps that make up the ISFS modeling process. They may also be invoked automatically by using processing scripts as explained in the subsections below.

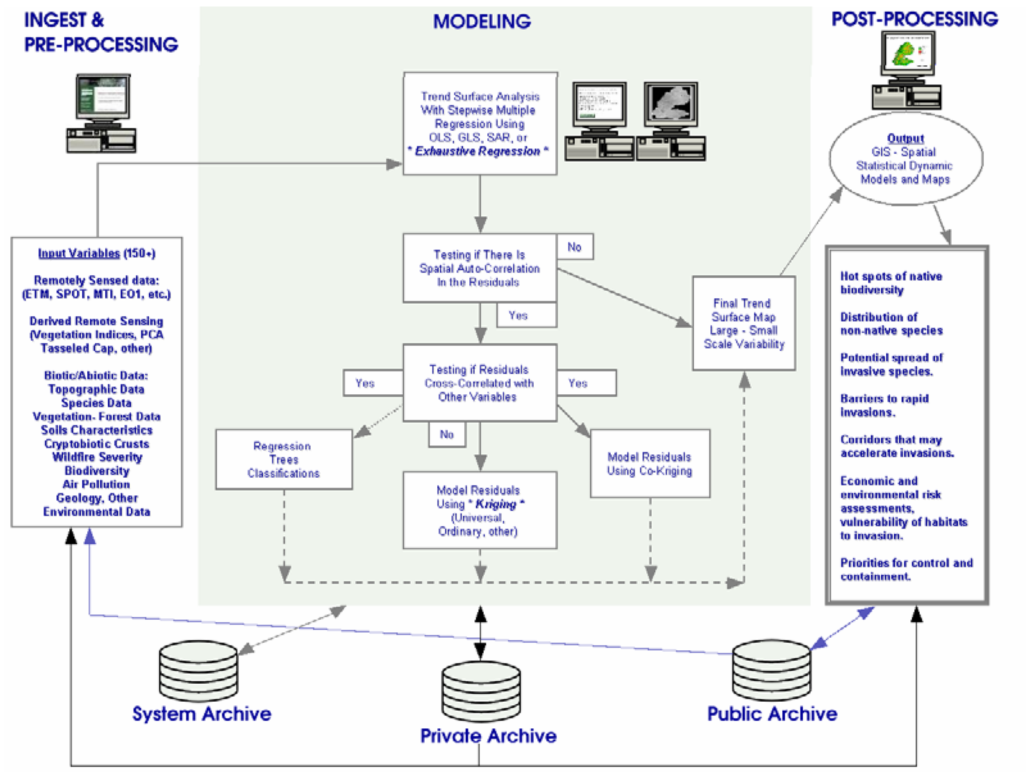

INGEST &
PRE-PROCESSING
MODELING
POST-PROCESSING
Trend Surface Analysis With Stepwise Multiple Regression Using OLS, GLS, SAR, or * Exhaustive Regression *
Output
GIS - Spatial Statistical Dynamic Models and Maps
Input Variables (150+)
Remotely Sensed data: (ETM, SPOT, MTI, EO1, etc.)
Derived Remote Sensing (Vegetation Indices, PCA Tasseled Cap, other)
Biotic/Abiotic Data: Topographic Data Species Data Vegetation- Forest Data Soils Characteristics Cryptobiotic Crusts Wildfire Severity Biodiversity Air Pollution Geology, Other Environmental Data
Testing if There Is Spatial Auto-Correlation In the Residuals
No
Yes
Final Trend Surface Map Large - Small Scale Variability
Hot spots of native biodiversity
Distribution of non-native species
Potential spread of invasive species.
Barriers to rapid invasions.
Corridors that may accelerate invasions.
Economic and environmental risk assessments, vulnerability of habitats to invasion.
Priorities for control and containment.
Testing if Residuals Cross-Correlated with Other Variables
Yes
Yes
No
Regression Trees Classifications
Model Residuals Using * Kriging * (Universal, Ordinary, other)
Model Residuals Using Co-Kriging
System Archive
Private Archive
Public Archive

Figure 3 – System Flow Chart

3.1.2.1 Ingest Subsystem

The ISFS ingest subsystem will serve as the initial "entry point" for all data used in the system. It will ingest various forms of input data and place them within the data archive. The data fall into the three main categories: field point measurements, imagery, and ancillary (GIS) layers. Common to the three categories is that all data ingested into the system will be associated with some geographic location. The ingest subsystem will serve as the initial "entry point" for all data used in the system. There will be a validation step to verify the basic integrity of the data before ingest.

The ingest design may change as work on the Invasive Species Data System (ISDS) is begun. The ISFS and the ISDS will be separate systems that will interface with each other. The exact specifications of the interface are unknown at this time. The ISDS will produce a merged data set that is passed to the ISFS. The corresponding raster layers will be placed in the archive in preparation of running models on the data within the merged data set. It is unclear at this time how the functionality will be broken down with regard to both systems. This document will be updated as the requirements and design of the ISDS evolve.

Within the subsystem, users will be able to upload field data in a tabular form using standard templates, or tools provided by the Invasive Species Forecasting System. All required fields will be captured and will be in an accessible database format. Satellite data (e.g. ETM+, MODIS) will be included primarily from external satellite data archives (e.g. DAACs, NOAA, USGS), but user-supplied satellite or airborne imagery may also be used. The primary source for ancillary layers (e.g. roads, water, governmental boundaries) will initially be USGS, however the ingest system will allow user-supplied ancillary layers to be incorporated into the system. Data ingested into the system fall into three categories:

1. A Tabular file (typically from field observations and containing latitude, longitude, date information, and observed values at the given point)

2. Raster layers (typically image data or GIS "grids" which extend over a large area and have one or more layers of information and can have different spatial resolution for the pixel size).

3. A boundary area specifying the region of interest (such as boundary of a national park or monument). The area can be defined by a vector GIS file, a binary raster image, or simply a list of bounding coordinates. The system will allow only one tabular file and one boundary area but multiple raster data files for each modeling scenario. While the upper right, upper left, lower right and lower left may be too close to distinguish with the given GPS accuracy (that is, they would all be reported as the same values), this format would allow larger plots to explicitly define the given area (as opposed to a single point) and thus allow more explicit extraction of the image data for an appropriate area. The format for the tabular data will be a comma delimited ASCII file containing the following elements for each ground observation (hard returns are given here for clarity in the document, the actual values would be on one line):
   - siteID,
   - upper left X, upper left Y, upper right X, upper right Y
   - lower left x, lower left y, lower right x, lower right y
   - large plot ID (if the site is part of a larger plot, if not fill with something like -999)
   - date of collection,
   - observed variable 1, ..., observed variable N

Mechanisms for data acquisitions will be secured ftp-push for any user supplied data or automated secured ftp pull from the external archives. Users will be issued usernames and will be authenticated through passwords. User activity and preferences will be logged and archived in the system.

A specific list of external archives and required data sets will be established and maintained as part of the ingest subsystem. The interfaces to these archives will be negotiated and a thorough understanding of source and target schema will be included in the interface agreement.

For establishing the baseline canonical example, we assumed that the datasets exist and are stored locally on our servers. Once the HPC technology has been applied and proven we can expand our ingest routines to include additional themes (ancillary layers) and interfaces with government and university databases. Formal procedures for ingest of these data will be documented as the data/source specific requirements become evident. As a general standard operating procedure, we will capture metadata and QA type data that will describe each file to be ingested and write that into the file header or store it in a database record associated with the unique file identifier.

3.1.2.2 Pre-processing Subsystem

The pre-processing stage manipulates the data resulting from the ingest stage into a format/structure that can be used by the modeling component of the system. Limited pre-processing is needed for the tabular data, since the format will be specified in the ingest phase.

The subsystem may perform resampling if the input raster layers are not at the same resolution. The merged data product will be written to the archive in a common analysis format, possibly GeoTIFF.

The primary component of the pre-processing stage will be to extract, for each site (as defined by the coordinates in the tabular file), the information from each raster layer for that site. These values extracted from the raster layers will be appended as columns to the ingested tabular file. The results of this pre-processing step will be referred to as the *"merged tabular data"*. A model builder selects items from the merged data set to build the model array.

A secondary component of the pre-processing stage will be to create new variables from existing columns of the merged tabular file. These new variables will either be pre-programmed or user-defined functions of the existing columns. The results of this step will be referred to as the *"merged, augmented tabular file"*. Either a "merged tabular file" or a "merged, augmented tabular file" is required to go on to the modeling stage.

3.1.2.3 Post-processing Subsystem

The post-processing subsystem uses outputs from the Modeling Subsystem to generate visual and graphical products that are then made available to the user. An example run would apply the empirical model formula resulting from an Ordinary Least Squares (OLS) regression to the input satellite and Digital Elevation Model (DEM) data, to provide a preliminary map of the total plants in the region. Kriged estimates of the residuals of this regression are then added to this map to produce an improved map. The post-processing subsystem will typically produce a standard, application interchangeable output file, such as an IDL file with ENVI header information, that can be used by other applications to reproject the data and overlay with other data layers as requested by the user. The final data products will be packaged with the appropriate metadata, assigned a unique data set identifier, and archived.

3.1.3 Back End Layer

The Back End layer acts as the compute server for performing model runs and serves as a data archive. These systems reside on the host of the Linux Beowulf cluster. The host node must have file system access to the data archive where the model input and output files are stored. There is a server that will communicate asynchronously with the application layer using either an RMI or SOAP protocol. It will contain a scheduler module to manage the model runs submitted to the Linux cluster.

3.1.3.1 Modeling

Modeling in the first version of the ISFS will be empirical in nature and utilize statistical techniques. The general theme of the modeling will be to predict a certain species migration through or invasion of habitat based on remote sensing imagery and ancillary data layers. The modeling subsystem will support five basic workflow activities, for which detail may be found in the Baseline Design Doc:

Data Array Construction – constructing the merged dataset (i.e. array) containing the response, or dependent, predictor variables and geographic coordinates

Model Selection and Fitting - model selection involving screening predictor variables to determine which are most related to the dependent variable. Stepwise regression is such a method.

Model Diagnostics – assessing how well the model fits spatially to the data.

Model Acceptance/Adjustment/Refinement - confirm the appropriateness of the model based on the diagnostic results.

Model Output – the model formula is presented to the user along with the merged dataset and a map.

Each of these steps may contain one or more algorithms. The majority of the algorithms are written in custom IDL code and using library routines from ENVI and the IDL Astronomy User's Library. If the user has requested kriging, the intermediary results from the IDL algorithms are copied to each of the cluster nodes and the kriging is performed on the cluster. At this time, kriging is the only CPU intensive modeling step that is parallelized on the Beowulf cluster.

3.1.3.2 Data Archival

The persistent storage for the input and output files to the model are also in the Back End layer. The ingest system will allow users to stage data files on the back end archive. The files will be placed into an on-line file system and registered within a relational database. The archive subsystem will be comprised of system, private and public partitions. The RDBMS will store pointers to archived files that are indexed with a unique file ID. For externally stored data, the archive system will store a file ID and pointer or URL that can be used to retrieve and stage the archived files for subsequent processing. The RDBMS will support queries and allow users to locate both the input and output files that were used for a specific model run. There will be an archive manager module that will manage disk space within the archive. The manager will record all data set creation, updates and deletions with the RDBMS registry. It will also remove expired data files. An API will be supported that will allow the user to access the archive manager and manage their own personal data.

3.2 User Roles

When a user logs into the application, a profile maintained in the database is used to authenticate the user and set system preferences. Activities are implemented by a library of routines accessed through the controls on a web based forms interface. Roles include the following:

- “Administrator” — This role provides complete access to the ISFS Web system. An administrator would be able to manage other users and have access to their workspace, preferences and authorization information.
- “Model Builder” — This role is intended for the user that wishes to configure and tailor the model related tasks. The Model Builder is someone who has been authenticated and authorized to upload and ingest data in preparation for other users to perform repeated model runs. The Model Builder is a user who has registered with the system and maintains an active logon name and password to access the system. The Model Builder maintains a profile within the system that remembers the Builder's data selections and makes those selections available to the Builder upon login.
- “Model User” — This role is intended for a user to select specific input data and run it through a previously configured ISFS model run. The Model User is a user who has registered with the system and maintains an active logon name and password to access the system. The Model User maintains a profile within the system that remembers the User's data selections and makes those selections available to the User upon login.

The roles will be maintained by expanding on the features of PostgreSQL database capabilities, or can incorporate the Lightweight Directory Access Protocol (LDAP). LDAP is a protocol for accessing online directory services over the TCP/IP network protocol, and can be used to access standalone LDAP directory services or directory services supporting the X.500 standard. It provides a standard way for Internet clients, applications and Web servers to access directory listings of thousands of Internet users. This listing may be found at: (http://wp.netscape.com/newsref/pr/newsrelease126.html).

3.3 Input/Output

The section will detail the various input data that is required for a successful modeling run and the output files that are produced by the run. Each major file is listed below, along with a brief description of its origin and how it is used.

3.3.1 Input Files

Merged Dataset
This is tabular dataset that contains field data along with the corresponding x and y geospatial values. These field positions are then merged with corresponding imagery data at the same locations producing a merged file of both field information and imagery information for each specified geospatial location.

Model Array
The model array is a subset of the merged dataset. A user may pick and choose columns from the merged dataset to create a model array that is customized for a particular model run. This file is output from the ingest system and used for input to the model. This dataset is archived for future reference.

Parameters XML file
An XML file is created by the application that contains various user defined modeling parameters and file names for a model run instance.

Remote Sensing Raster Data
For each predictor variable in the model array, a corresponding raster file must exist in the archive. Therefore if a model array contains n columns of remote sensing data values, there must be n data files available that correspond to each of the columns. The regression model that is calculated based on the model array must be applied to the corresponding raster data to obtain a resultant landscape image.

3.3.2 Output Files

GeoTiff

Based on the calculated statistical model, a landscape image is produced of the response variable. The file produced will be a GeoTiff that is easily imported into many standard GIS tools.

JPG

The model will also produce a “quick look” browse image of the GeoTiff, viewable in the browser.

Model Run Parameters Metadata File

The empirical model will be saved along with various input and derived parameters. This includes such things as the type of statistical model that was used, all the pertinent coefficients and the algorithms chosen by the user. This file should include all the information required to recreate the results.

3.4 System Operations

3.4.1 Restrictions

Users will be restricted to various resources within the forecasting system depending on their role and the availability of the various resources. One aspect of user roles is that it will allow the system to wisely grant access and manage resources based on the user's need. For instance, model builders will have more access to archive and CPU resources.

On the back end, the CPU, archive disk space, file system and job scheduling will all be managed to optimize response and throughput. Job scheduling will be required to assure that each user will have access to the cluster with reasonable turn-around time. Initially the scheduler may be implemented as a FIFO. Each job is run sequentially in the order it was submitted. Later, the scheduler will include logic that better manages the nodes and archive resources.

The model algorithms and source tree will only be available to the model builders and the system administrators. Model users will not be able to upload new algorithms to try, they will be restricted to the use of the existing algorithm tool box.

Archive data space will be restricted to three main repositories that are managed independently. The public archive will contain an area where registered model users may store data that is staged for model runs along with model run results. The private archive will be restricted to model builders. These data will not be available to the general registered user. There is also a system archive where canonical and ancillary data are stored.

3.4.2 Compute Server

The Beowulf cluster will be used as the principle compute server when performing a model run. The Application layer packages a model run request and passes it to the compute server on the back end. The request will be an XML file that includes all input file names and input parameter values along with other pertinent descriptive fields (see example in Input/Output Samples appendix). The model run begins by calculating a distance matrix for all the field samples. A bounding box is then determined that includes all the field samples contained in the merged dataset. A regression model or some other statistical model is then applied to the tabular data to derive a statistical model. The next step is to create a variogram to look for un-modeled spatial information in the residuals. The steps up to this point are relatively light weight and thus are performed on only one cluster node. They are implemented in IDL to take advantage of IDL/ENVI libraries and to facilitate rapid prototype of this dynamic part of the application.

The next step in the modeling flow is to perform kriging. Kriging is extremely processor intensive and for this reason the kriging code will be parallelized and submitted to run on the Linux cluster. Once the kriging has completed, the results files are created and placed in the archive where they are available for display to the user.

3.4.3 Operational Scenario

The following diagram illustrates the current scenario of invoking a model run using ISFS. The current protocol is based on SSH, since port 22 is the only way to access resources behind the firewall. This limitation has forced the prototype to be built around SSH and SCP. A more robust protocol between the application and back end, such as RMI or SOAP, will be required before moving beyond the prototype functionality. In the figure below, the application, running on the web server WEBSERV, uses SCP to copy the merged data set file.modelarray to the back end host server FRIO. In addition, an XML file file_param.xml containing pointers to data files and model parameters is passed to FRIO. The model is then invoked by executing isfs.sh and execution begins. At this point IDL and ENVI are launched, reading the XML file that includes the user configuration. WEBSERV polls FRIO using SCP until the model has completed and the output file is generated. The output file is then copied to WEBSERV and displayed for the user. Note that the need to poll for model results is a result of using SSH as the transport between the application and back end layers. Once an RMI or SOAP interface is implemented, the model results will be returned asynchronously, thus eliminating the need for the polling.

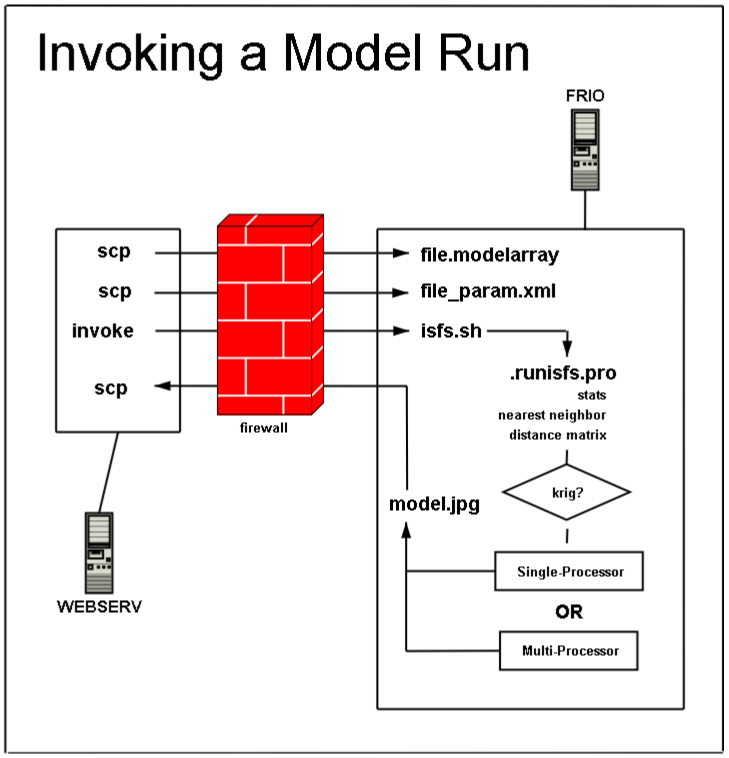
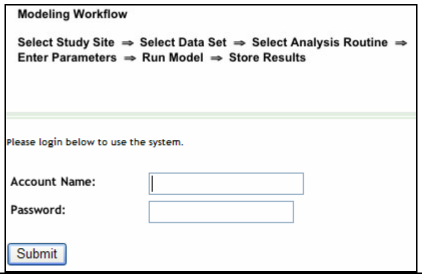
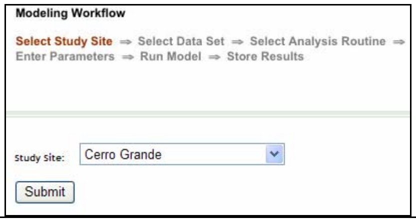
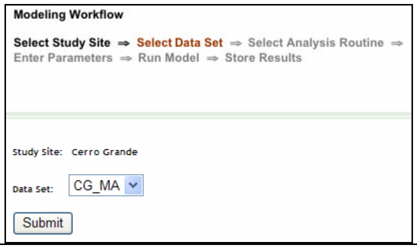
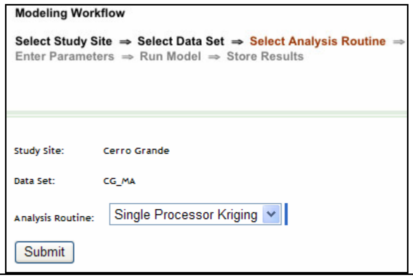
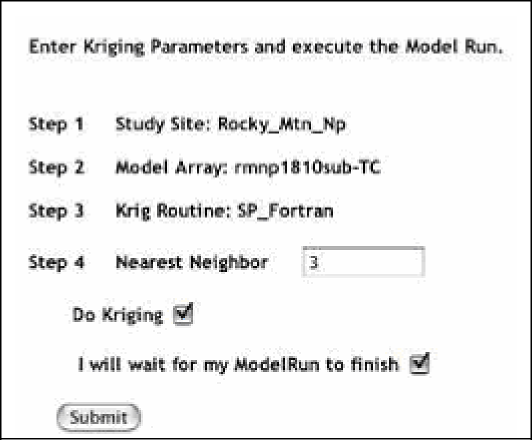
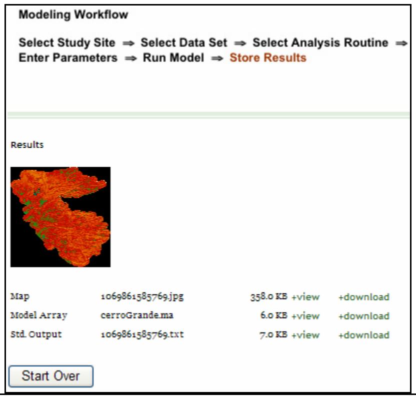
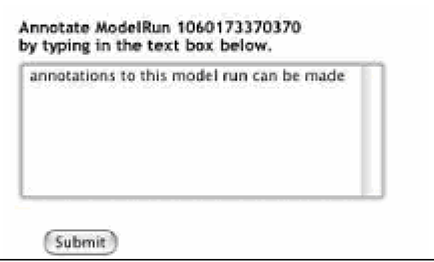


Figure 4 - The web application interfaces with the back-end layer which resides behind a firewall.

3.4.4 Model Run Use Case

The following use case describes the steps a user would make in the application and how each request is processed in order to run a model. The application GUI presented in the browser, consists of screens created & served by java server pages via the Tomcat servlet engine. The following diagram is based on "*Run a Pre-defined Model*" Use Case found in the Concept of Operations document.

Step	GUI	System Response
1. User enters account name and password to enter the system.	 Modeling Workflow Select Study Site ⇒ Select Data Set ⇒ Select Analysis Routine ⇒ Enter Parameters ⇒ Run Model ⇒ Store Results Please login below to use the system. Account Name: Password: Submit	• User is authenticated in lookup table and enters the system • The system locates list of Study Sites from the DB that are ready to be submitted as a model run. • System locates list of kriging types from DB . • Using JSP, page is drawn with list of study sites. • If the user is not authenticated, they are brought back to the login page with the JSP displaying an error message that reads *"Your Account name or password is invalid"*.
2. User selects study site and submits selection	 Modeling Workflow Select Study Site ⇒ Select Data Set ⇒ Select Analysis Routine ⇒ Enter Parameters ⇒ Run Model ⇒ Store Results Study Site: Cerro Grande Submit	• System locates list of Model Arrays (MA) from DB • Using JSP, draw page with Model Array list for the selected study site
3. User selects model array and submits selection	 Modeling Workflow Select Study Site ⇒ Select Data Set ⇒ Select Analysis Routine ⇒ Enter Parameters ⇒ Run Model ⇒ Store Results Study Site: Cerro Grande Data Set: CG_MA Submit	• System locates list of Kriging routines from DB • Using JSP, draw page with Kriging routines list for the selected study site
4. User selects Kriging routine and submits selection	 Modeling Workflow Select Study Site ⇒ Select Data Set ⇒ Select Analysis Routine ⇒ Enter Parameters ⇒ Run Model ⇒ Store Results Study Site: Cerro Grande Data Set: CG_MA Analysis Routine: Single Processor Kriging Submit	• Using JSP, draw page with Nearest Neighbors (NN) field

5. User enters NN value and submits	 Enter Kriging Parameters and execute the Model Run. Step 1 Study Site: Rocky_Mtn_Np Step 2 Model Array: rmnp1810sub-TC Step 3 Krig Routine: SP_Fortran Step 4 Nearest Neighbor 3 Do Kriging ✓ I will wait for my ModelRun to finish ✓ Submit	• System validates all input: (a) Files exists MA (and possibly kriging) (b) Limit checks on NN (c) Cluster Ready/Available (d) Has this setup already been run? • Send MA, krig type, NN and a process identifier to cluster host (compute server) and invoke isfs.sh as `sh isfs.sh unique_id' • The model runs and produces output in `/from../procid/outputfile.jpg,` [as well as a GeoTiff version]. Please see Input/Output Samples section for examples and description of the model output. • When the model run is complete, a java program executes and places an RMI call back to the controller passing the .jpg, procid and associated metadata (an XML file) back to the controller. • The controller is notified and stores the map image on the file system. The metadata is entered in the DB and the image is displayed in the browser. The metadata for the run is displayed. The user is allowed to add comments to a field within the metadata record
6. User views large map image by selecting 'download' next to Map output item	 Modeling Workflow Select Study Site ⇒ Select Data Set ⇒ Select Analysis Routine ⇒ Enter Parameters ⇒ Run Model ⇒ Store Results Results Map 106986158576 9.jpg 358.0 KB +view +download Model Array cerroGrande.ma 6.0 KB +view +download Std. Output 106986158576 9.txt 7.0 KB +view +download Start Over	• Large image of the thumbnail of the map is displayed in the browser
7. User chooses to save the model run & annotates notes	 Annotate ModelRun 1060173370370 by typing in the text box below. annotations to this model run can be made Submit	• This is not prototyped, but user will have the ability to store the resulting model formula, standard output, map and associate notes to that persisted data.

4 Object-Oriented Design

4.1 Introduction

ISFS incorporates Java Server Pages within the web application layer to make database calls and activate the model application on the back-end layer. The web app logic is implemented in Java and run using Jakarta Tomcat as the servlet engine. Jakarta Struts was chosen to act as an extensible framework within Tomcat. The Struts framework allows for a clean distinction between the presentation layer in JSP and the business logic, and thus encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm.

When an HTML form is filled in by the user, it is passed to a Struts servlet that acts as a controller. The controller will then dispatch a request to the appropriate Action class based on the object that is passed. The advantages are that the presentation code (JSP or HTML) is not embedded within servlet code, and each action is functionally distinct, creating a lower maintenance and a more cohesive application that is easier to extend than otherwise possible. The Action class is then responsible to carry out the specified functionality.

Struts provides its own Controller component, and integrates with other technologies to provide the Model and the View. For the Model, Struts can interact with any standard data access technology, including Enterprise Java Beans, JDBC, and Object Relational Bridge. For the View, Struts works well with JavaServer Pages, including JSTL and JSF, as well as Velocity Templates, XSLT, and other presentation systems.

The Struts framework provides the invisible underpinnings that a professional web application needs to survive. Struts helps to create an extensible development environment for the application, based on published standards and proven design patterns. Struts is part of the Apache Jakarta Project, sponsored by the Apache Software Foundation. Additional information is available from the official Struts home page http://jakarta.apache.org/struts.

The following sections describe the Struts construction of the ISFS application in terms of Model Run Design, User Accounts Design, ISFS Common Design, Package and Class groups. These models and diagrams form the basis for ISFS application development.

4.2 Packaging

The classes are packaged into a top level 'isfs' package (see Figure 5 below). Within the isfs package there is a package 'accounts' for managing user accounts and sessions, there is a common package 'common' of utility type classes, there is a 'modelrun' package the contains the classes needed to configure and run a model instance, an action package to hold the Struts actions and a screens to contain the JSP.

Within the ‘accounts’ and ‘modelrun’ packages there are four additional sub-packages. 'app' contains the application logic within the package of classes, 'bus' which contains the classes relating to the business logic that controls and manages the data objects, 'valueObj' that contains the Javabeans that the JSP pages require and finally 'db' deals with the logic involved with interacting with the database to store and restore value objects. The db package contains DAO objects for all of the classes in the valueObj package. The Javadoc for the packages and classes may be referenced on-line at http://carbon.sesda.com/bp/intranet/html_docs . (note this is password protected)

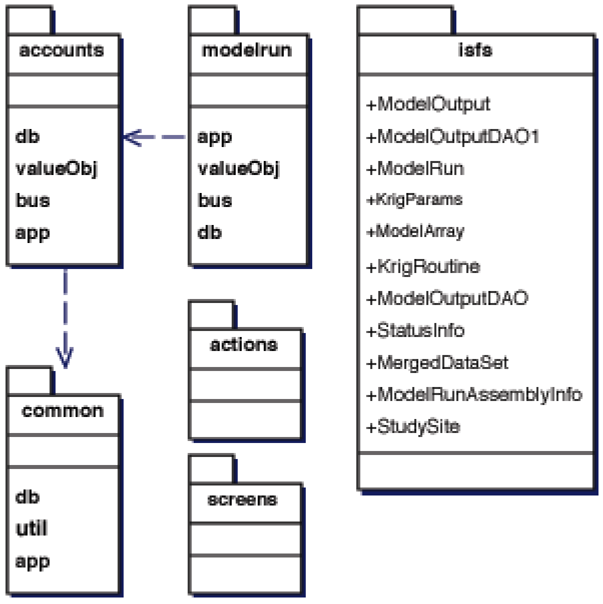


Figure 5 - ISFS Packages

The following sections will give a brief description of the existing classes within the major packages. The text and the diagrams are based on the current working prototype. It is in no way complete and thus the class diagrams are not complex. There is only one example of inheritance and a limited number of associations between classes. This results in a number of isolated classes. As the system design matures and functionality is extended, the complexity of the classes and their inter-relationships will increase. The class diagrams were produced using Borland Together 6.1.

4.3 Model Run Design

4.3.1 Overview and Responsibilities

The ISFS modelrun package contains classes that encapsulate the ModelRun concepts, and implement the execution, and editing of ModelRun objects. Like all of the other packages it is broken up into four sub-packages, app, bus, db and valueObj. The main interface for ModelRun activity is the ModelRunService class, which gets constructed and placed into the session when a user logs into the system. From there, the ModelRunService is used by the app classes for all of the ModelRun assembly, execution and editing processes.

Two UML run sequence diagrams involving the launch and processing of a model run are explained here and located in the Model Run Class Diagrams section below. A class diagram and more component series charts are also included, which present the relationships & interaction of objects at various contexts of a model run.

Run Model Sequence Level 1

In Figure 7, the sequence diagram illustrates the chronological steps resulting from a user request to configure & execute a model. Objects are enumerated horizontally by rectangles at the top of the diagram. Arrows between objects indicate an access or method invocation between objects.

Run Model Sequence Level 2

Once input parameters have been extracted and validated from user input, this sequence is executed to launch the model run and wait for results. There is a fixed wait period, as the current architecture does not permit the server to control the browser. Therefore, client-side scripting is necessary to poll the system, and is executed by the browser. This sequence, shown in Figure 8 is nested within Run Model Sequence Level 1.

4.3.2 Class Descriptions

bus package

ModelRunService: class used to access all information needed to alter,access and execute ModelRuns. Serves as the interface for the application classes.
SecureCopyThread: Implementation of SCP used to securely copy files to and from the ISFS ModelRun kriging server.
ModelRunController: controller class that serves as the mechanism for submitting ModelRun jobs. Currently, this class holds a queue of jobs and executes each job via SSH in the order that the jobs were submitted. In the future, the plan for this class would be to communicate with the ModelRunServer via RMI for job submission and the ModelRunServer would handle each individual job.
ModelRunServer: Not Implemented, described briefly above.
MyUserInfo: Class needed by the SSH and SCP classes to give user information.

app package

BuildModelRunAction: implementation of the Struts DispatchAction class which contains methods for the sequence of assembling a ModelRun and having it execute.
BuildModelRunActionForm: implementation of the Struts ActionForm class which is populated across a multiple pages, once the form is completed its values are used to execute a ModelRun.
AnnotateModelActionForm: implementation of the Struts ActionForm class, which is filled in during the annotation of a ModelRun. Its data is eventually stored in the database as an annotation to the modelrun.
EditModelRunAction: implementation of the Struts DispatchAction class that contains methods for editing and viewing ModelRuns.

valueObj package

ModelRun: the encapsulation of a ModelRun, which contains all of the assembly information such as study site, modelarray, etc., that is needed to execute a model run . ModelRuns are created before they are run so some ModelRuns have no ModelOutput yet.
ModelOutput: encapsulated results of a ModelRun.
MergedDataSet: encapsulation of a mergeddataset file.
StudySite: describes a site from which data and images were drawn.
KrigParams: parameters class encapsulating the parameters of a KrigRoutine.
ModelArray: encapsulates the model array data file used as the data in a ModelRun.
ModelRunAssemblyInfo: convenient bean used to store all of the possible combinations of data that could be used to assemble a ModelRun. This bean is filled when a user starts to assemble a ModelRun and is stored in the session along with the ModelRunService, so that the system does not have to repeatedly access the database to get this information.
KrigRoutine: representation of a KrigRoutine which contains attributes to describe itself and its possible parameters along with the location from which it can be used.
StatusInfo: bean placed in the session when a ModelRun is submitted, which keeps an attribute that is the percentage of completion for the ModelRun submitted in this session.

4.3.3 Model Run Class Diagrams

The figures below depict the class & sequence diagrams related to the ModelRun classes/objects.

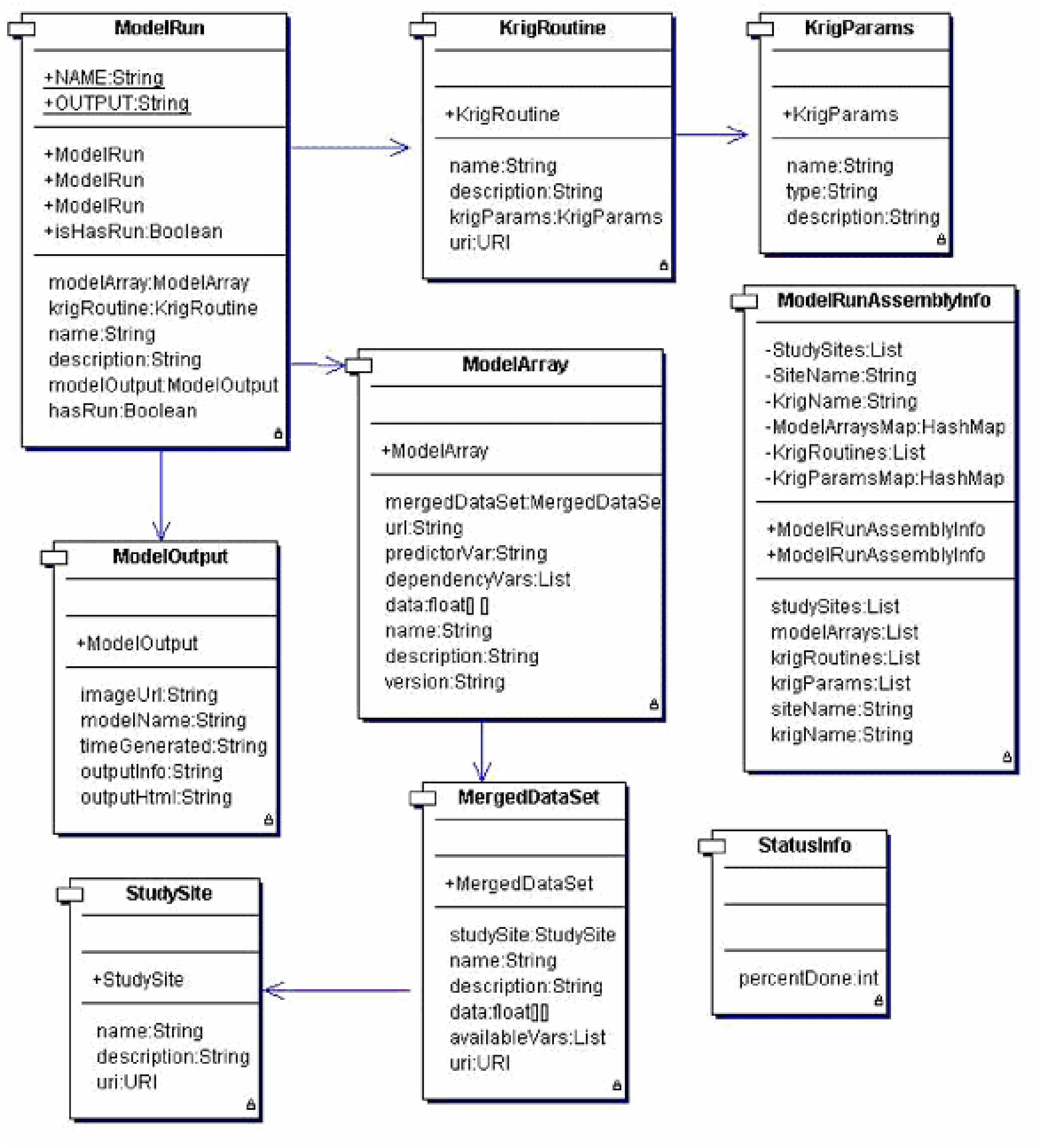


Figure 6 - ModelRun\valueObj class diagram as generated by TogetherJ UML/IDE Tool

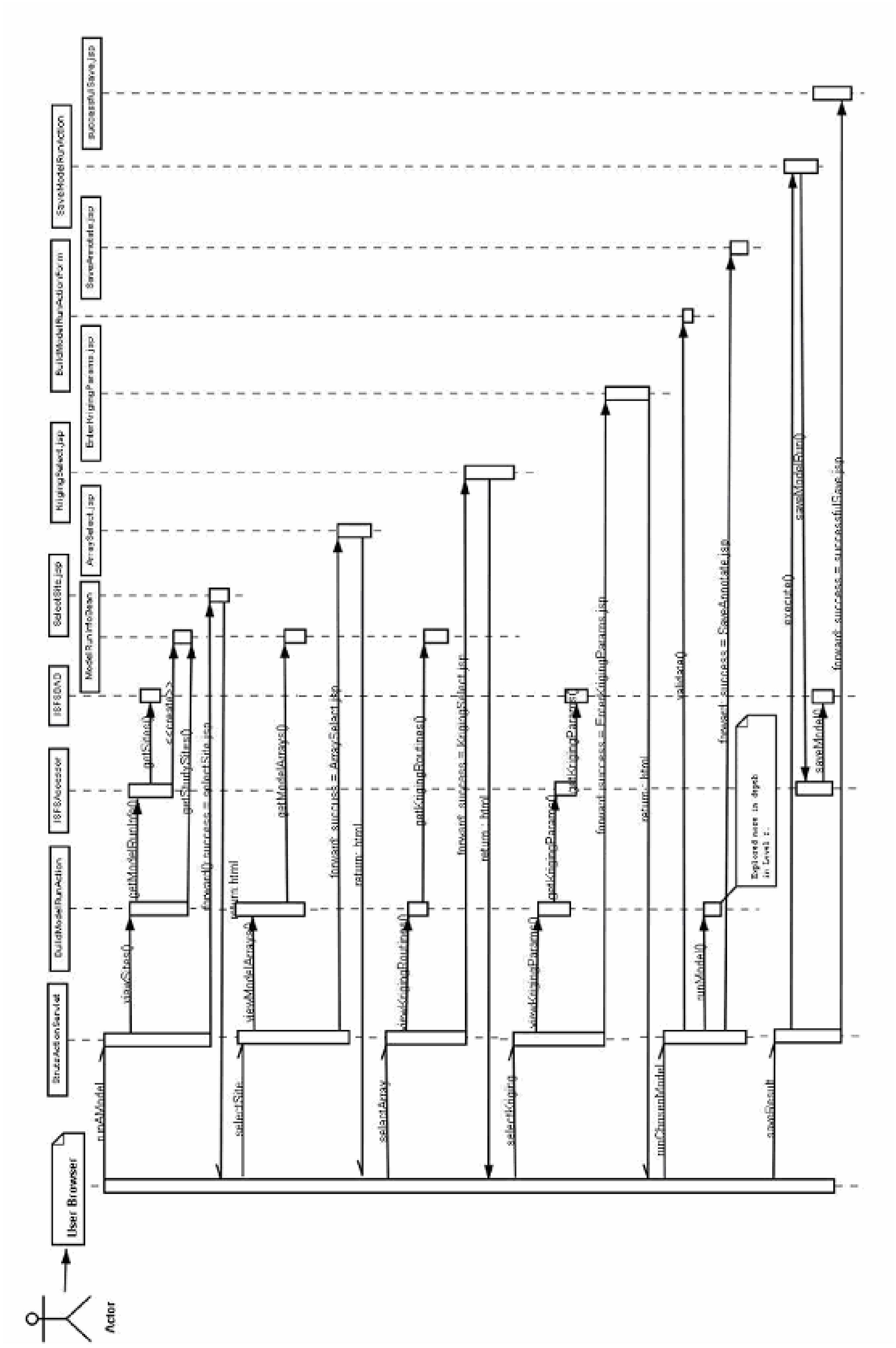


Figure 7 - Model Run Sequence Level 1

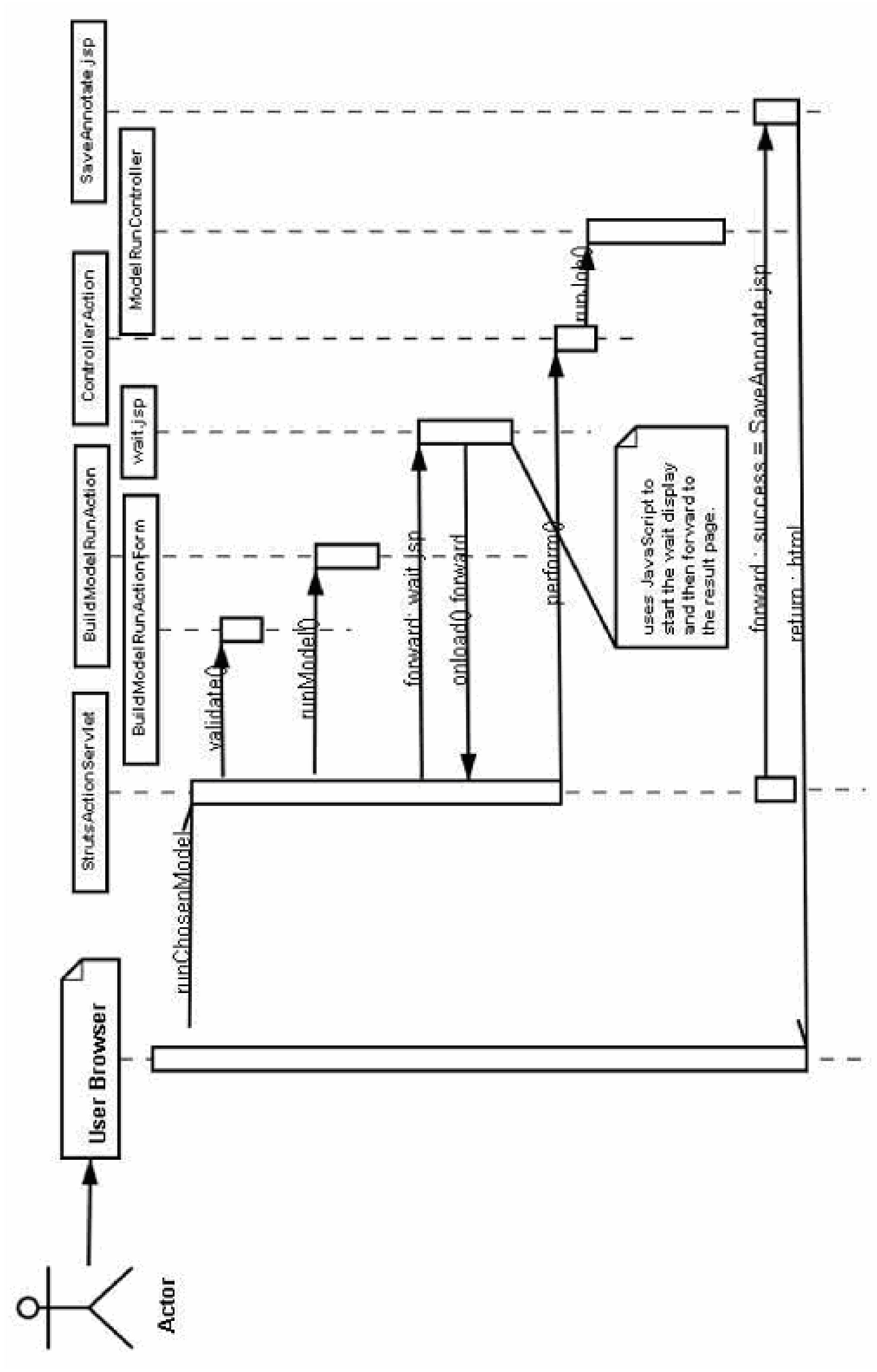


Figure 8 - Model Run Sequence 2

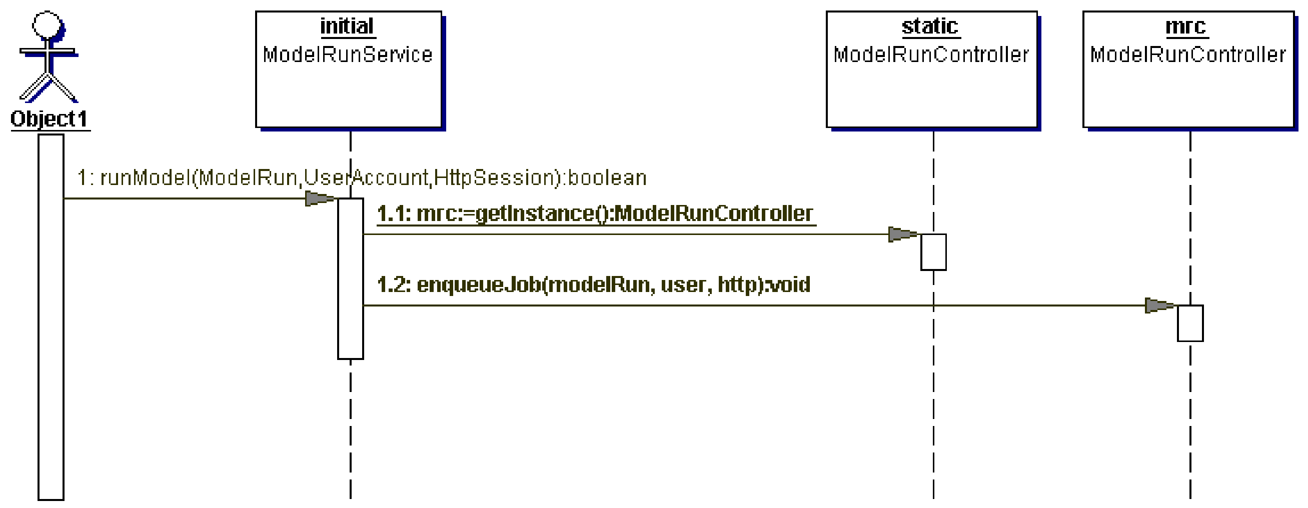

Object1
initial
ModelRunService
static
ModelRunController
mrc
ModelRunController
1: runModel(ModelRun,UserAccount,HttpSession):boolean
1.1: mrc:=getInstance():ModelRunController
1.2: enqueueJob(modelRun, user, http):void

Figure 9 - Sequence diagram of the runModel method of the BuildModelRunAction class, which places the job into the ModelRunControllers job queue.

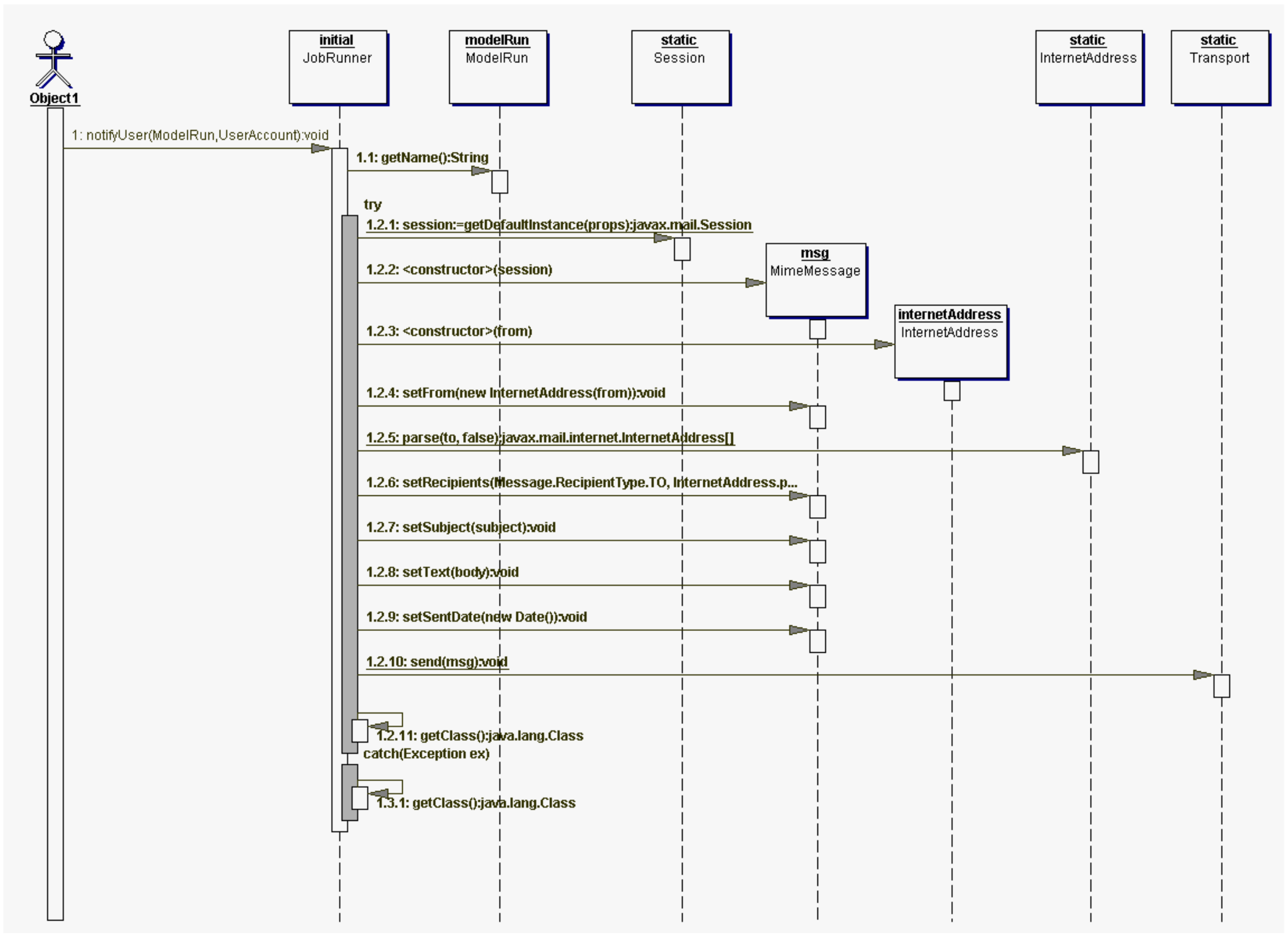

Object1
initial
JobRunner
modelRun
ModelRun
static
Session
static
InternetAddress
static
Transport
msg
MimeMessage
internetAddress
InternetAddress
1: notifyUser(ModelRun,UserAccount):void
1.1: getName():String
try
1.2.1: session:=getDefaultInstance(props):javax.mail.Session
1.2.2: <constructor>(session)
1.2.3: <constructor>(from)
1.2.4: setFrom(new InternetAddress(from)):void
1.2.5: parse(to, false):javax.mail.internet.InternetAddress[]
1.2.6: setRecipients(Message.RecipientType.TO, InternetAddress.p...
1.2.7: setSubject(subject):void
1.2.8: setText(body):void
1.2.9: setSentDate(new Date()):void
1.2.10: send(msg):void
1.2.11: getClass():java.lang.Class
catch(Exception ex)
1.3.1: getClass():java.lang.Class

Figure 10 - Sequence diagram of the ModelRunController, which notifies the user when a ModelRun is complete

4.4 User Accounts Design

4.4.1 Overview and Responsibilites

The ISFS ModelRun tool is intended to be used by authorized users only. Currently there is only one type of user that may execute, view and annotate ModelRuns. In future releases, this will be expanded to add multiple levels of users and user capabilities that will include roles for administrators, model builders & model users. Security measures for user passwords and authorization levels will also be implemented.

The UserAccountsManager should eventually have the following responsibilities:

- provide an interface for the user access to the system;
- provide information about individual users;
- encrypt and decrypt user passwords;
- update user information; and
- create and remove users for the system

4.4.2 Class Descriptions

bus package
UserAccountsManager: the primary class with which the Presentation Layer interfaces. The UserAccountsManager provides access to all user operations.

valueObj package
UserAccount: encapsulates ISFS user information. This object is created when a user logs onto the system and is placed in the users http session.

app package
LogonAction: the struts action used to logon users.
LogonActionForm: the Struts ActionForm for the logon process.

dbPackage
UserAccountDAO: the database access object for user accounts. This class knows how to access all of the user information stored in the database and how to construct a UserAccount from the stored data.

4.4.3 User Account Class Diagrams

The figures below depict the class & sequence diagrams related to the UserAccount classes/objects.

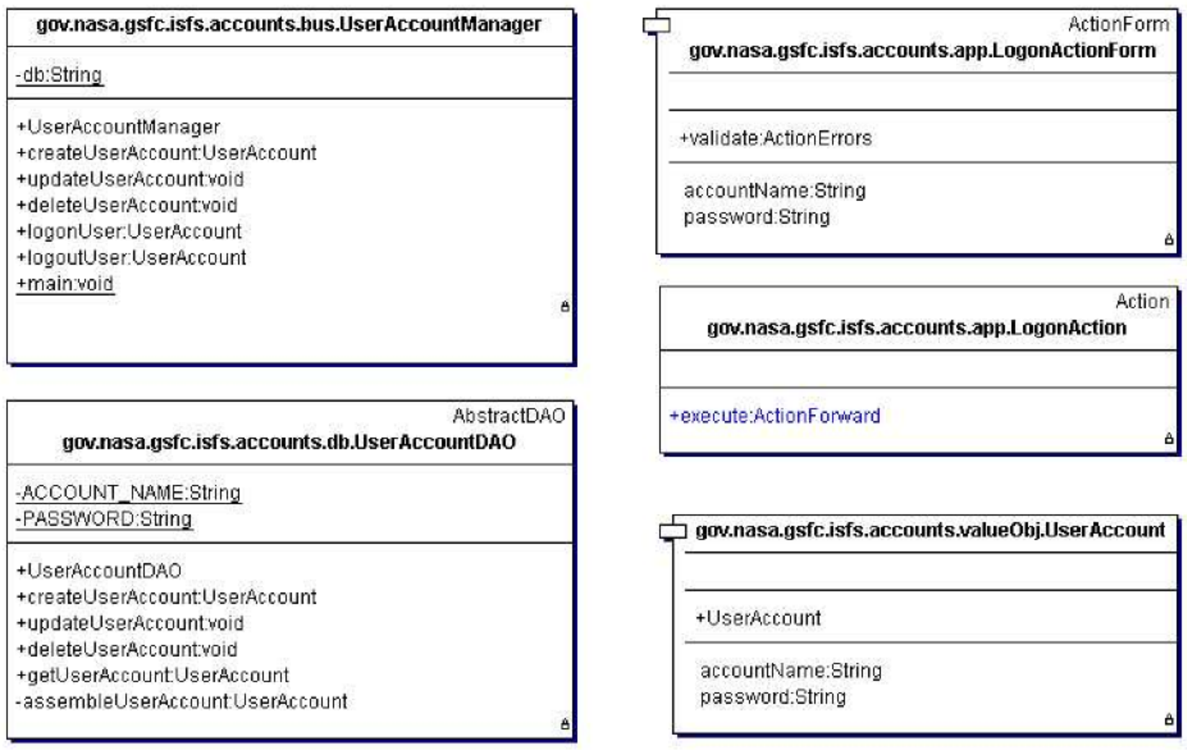


Figure 11 - Accounts Package class diagram as generated by TogetherJ UML/IDE Tool

4.5 ISFS Common Design

4.5.1 Overview and Responsibilities

The ISFS common package contains all of the shared database and utilities classes needed for the ISFS ModelRun tool. This is the package where we place all of the utilities and classes that do not fit well within other packages.

4.5.2 Class Descriptions

AbstractDAO
This class handles all of the generic DAO JDBC functionality. It is intended to be extended with specific functionality for the object that needs DAO support.

ISFSDAO
This class is intended to have more specific DAO functionality for the ISFS project. Many of the ISFS DAO objects extend this class.

Transaction
Encapsulates a database transaction and is used within the AbstractDAO class.

ISFSDbException:
ISFS Exception wrapper class around the JDBC database exception. Logger: ISFS wrapper class around the Log4j logging.

4.5.3 Utility Diagrams

The figures below depict the class & sequence diagrams related to the Common classes/objects.

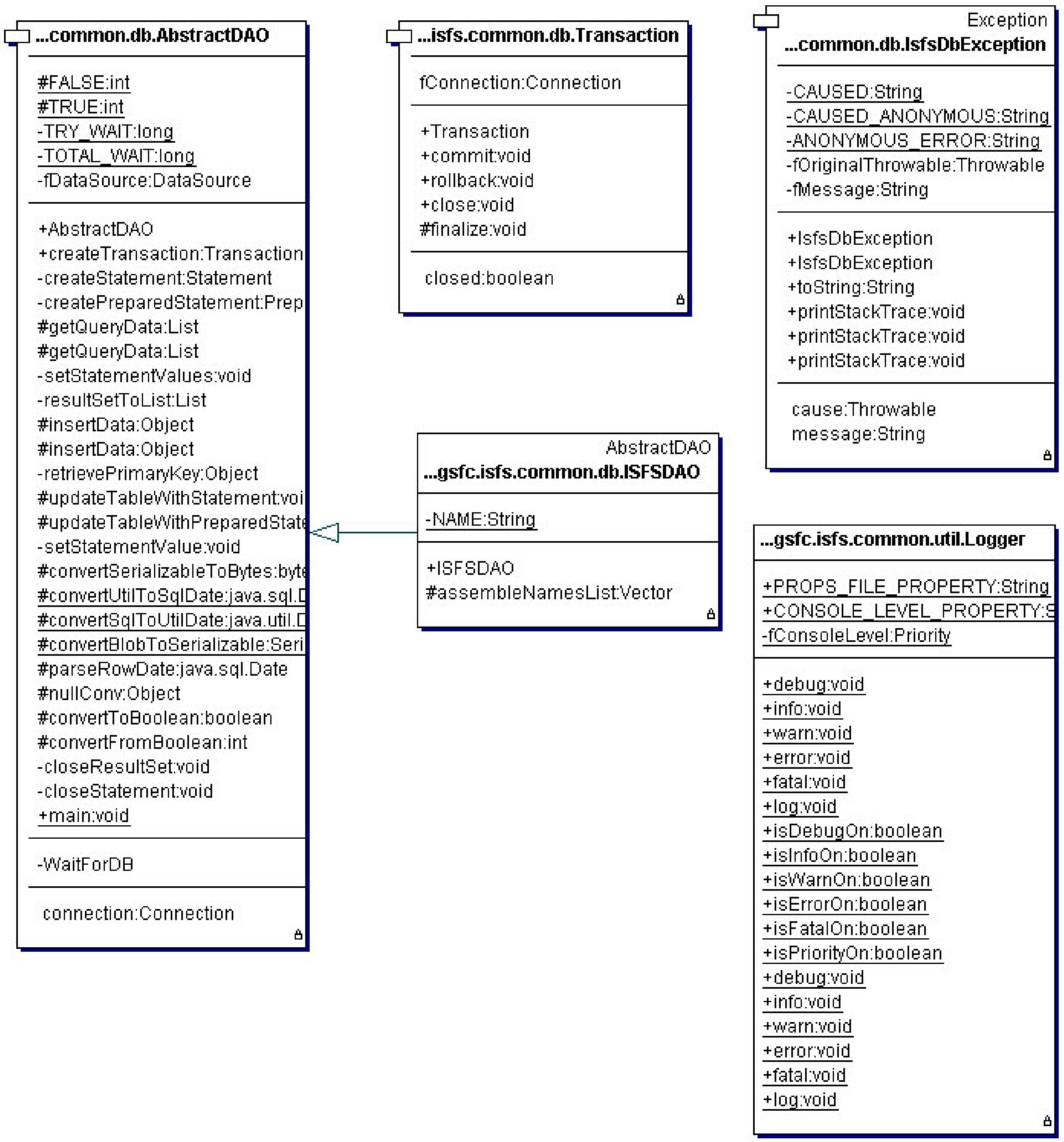


Figure 12 - Common package class diagram as generated by TogetherJ UML/IDE Tool.

4.6 Hierarchy For All Packages

<default>, design, design.doc-files, gov, gov.doc-files, gov.nasa, gov.nasa.doc-files, gov.nasa.gsfc, gov.nasa.gsfc.doc-files, gov.nasa.gsfc.isfs, gov.nasa.gsfc.isfs.accounts, gov.nasa.gsfc.isfs.accounts.app, gov.nasa.gsfc.isfs.accounts.app.class-use, gov.nasa.gsfc.isfs.accounts.app.doc-files, gov.nasa.gsfc.isfs.accounts.bus, gov.nasa.gsfc.isfs.accounts.bus.class-use, gov.nasa.gsfc.isfs.accounts.bus.doc-files, gov.nasa.gsfc.isfs.accounts.db, gov.nasa.gsfc.isfs.accounts.db.class-use, gov.nasa.gsfc.isfs.accounts.db.doc-files, gov.nasa.gsfc.isfs.accounts.doc-files, gov.nasa.gsfc.isfs.accounts.valueObj, gov.nasa.gsfc.isfs.accounts.valueObj.class-use, gov.nasa.gsfc.isfs.accounts.valueObj.doc-files, gov.nasa.gsfc.isfs.common, gov.nasa.gsfc.isfs.common.db, gov.nasa.gsfc.isfs.common.db.class-use, gov.nasa.gsfc.isfs.common.db.doc-files, gov.nasa.gsfc.isfs.common.doc-files, gov.nasa.gsfc.isfs.common.util, gov.nasa.gsfc.isfs.common.util.class-use, gov.nasa.gsfc.isfs.common.util.doc-files, gov.nasa.gsfc.isfs.doc-files, gov.nasa.gsfc.isfs.modelrun, gov.nasa.gsfc.isfs.modelrun.app, gov.nasa.gsfc.isfs.modelrun.app.class-use, gov.nasa.gsfc.isfs.modelrun.app.doc-files, gov.nasa.gsfc.isfs.modelrun.bus, gov.nasa.gsfc.isfs.modelrun.bus.class-use, gov.nasa.gsfc.isfs.modelrun.bus.doc-files, gov.nasa.gsfc.isfs.modelrun.db, gov.nasa.gsfc.isfs.modelrun.db.class-use, gov.nasa.gsfc.isfs.modelrun.db.doc-files, gov.nasa.gsfc.isfs.modelrun.doc-files, gov.nasa.gsfc.isfs.modelrun.valueObj, gov.nasa.gsfc.isfs.modelrun.valueObj.class-use, gov.nasa.gsfc.isfs.modelrun.valueObj.doc-files, gov.nasa.www, gov.nasa.www.doc-files, html docs

4.7 Class Hierarchy

- class gov.nasa.gsfc.isfs.common.db. AbstractDAO
    - – class gov.nasa.gsfc.isfs.common.db. ISFSDAO
        - class gov.nasa.gsfc.isfs.modelrun.db. KrigInfoDAO
        - class gov.nasa.gsfc.isfs.modelrun.db. MergedDataSetDAO
        - class gov.nasa.gsfc.isfs.modelrun.db. ModelArrayDAO
        - class gov.nasa.gsfc.isfs.modelrun.db. ModelOutputDAO
        - class gov.nasa.gsfc.isfs.modelrun.db. ModelRunDAO
        - class gov.nasa.gsfc.isfs.modelrun.db. StudySiteDAO
    - – class gov.nasa.gsfc.isfs.accounts.db. UserAccountDAO
- class gov.nasa.gsfc.isfs.modelrun.valueObj. KrigParams
- class gov.nasa.gsfc.isfs.modelrun.valueObj. KrigRoutine
- class gov.nasa.gsfc.isfs.common.util. Logger
- class gov.nasa.gsfc.isfs.modelrun.valueObj. MergedDataSet
- class gov.nasa.gsfc.isfs.modelrun.valueObj. ModelArray
- class gov.nasa.gsfc.isfs.modelrun.valueObj. ModelOutput
- class gov.nasa.gsfc.isfs.modelrun.valueObj. ModelRun
- class gov.nasa.gsfc.isfs.modelrun.valueObj. ModelRunAssemblyInfo
- class gov.nasa.gsfc.isfs.modelrun.bus. ModelRunController
- class gov.nasa.gsfc.isfs.modelrun.bus. ModelRunServer
- class gov.nasa.gsfc.isfs.modelrun.bus. ModelRunService
- class gov.nasa.gsfc.isfs.modelrun.bus. MyUserInfo (implements UserInfo)
- class java.lang.Object
    - – class org.apache.struts.action.Action
        - class org.apache.struts.actions.DispatchAction
            - class gov.nasa.gsfc.isfs.modelrun.app. BuildModelRunAction
            - class gov.nasa.gsfc.isfs.modelrun.app. EditModelRunAction
        - class gov.nasa.gsfc.isfs.accounts.app. LogonAction
    - – class org.apache.struts.action.ActionForm
        - class gov.nasa.gsfc.isfs.modelrun.app. AnnotateModelActionForm
        - class gov.nasa.gsfc.isfs.modelrun.app. BuildModelRunActionForm
        - class gov.nasa.gsfc.isfs.accounts.app. LogonActionForm
    - – class java.lang.Thread
        - class gov.nasa.gsfc.isfs.modelrun.bus. SecureCopyThread
    - – class java.lang.Throwable
        - class java.lang.Exception
            - class gov.nasa.gsfc.isfs.common.db. IsfsDbException
- class gov.nasa.gsfc.isfs.modelrun.valueObj. StatusInfo
- class gov.nasa.gsfc.isfs.modelrun.valueObj. StudySite
- class gov.nasa.gsfc.isfs.common.db. Transaction
- class gov.nasa.gsfc.isfs.accounts.valueObj. UserAccount
- class gov.nasa.gsfc.isfs.accounts.bus. UserAccountManager

5 Data Dictionary

A key part of the ISFS is the ability to store and persist various business classes and their state information between user sessions. User session data and model related metadata are stored within a relational database and accessed using JDBC. JDBC allows for a level of abstraction so the system is not dependant on a particular vendor's RDBMS. This database will also serve as a registry for the data archive on the back end. Each registered user will be associated with the various model runs s/he has executed. Likewise each of these model runs will contain model related metadata and pointers to the archive where the relevant input and output files for each run are stored.

The following tables enumerate the current data tables used within the working prototype database schema. The tables are not complete at this time and the schema will be extended and adapted as the project evolves. Each table is listed along with a column for each of its attributes. Following each table is a sample entry. The purpose of this Data Dictionary is to describe:

1. The function of each table
2. The attributes within each table including their modifiers and constraints
3. The relationships and dependencies of the tables.

5.1 Relational Data Tables

5.1.1 krigparams

A table used to support configurable kriging parameters. Krig routines may potentially use many different parameters. This table contains data that specifies possible parameters used by kriging algorithms. It is used by the web application to display parameter setting to the user for their input.

Table Definition:

Column Name	**Type**	**Modifiers**	**Description**
name	varchar	not null	The name for this parameter.
type	varchar	not null	The IDL data type of the parameter.
description	text		An optional meaningful description of this parameter.

Example Record:

Column Name	**Example**
name	Nearest Neighbor
type	Integer
description	NN

5.1.2 krigroutine

Table used to support definitions for multiple kriging algorithms/routines. Users may choose one of the kriging algorithms in this table for each model run.

Table Definition:

Column Name	**Type**	**Modifiers**	**Description**
name	varchar	not null	The name of the krig routine.
krigparamsname	varchar	not null	A parameter used by the kriging algorithm. The foreign key for a row in the krigparams table. Currently we only support one kriging parameter.
uri	varchar	not null	Currently not used.
description	text		An optional detailed description of this kriging algorithm.

Example Record:

Column Name	Example SP_Fortran	Example MP_Fortran	Example none
name	SP_Fortran	MP_Fortran	none
krigparamsname	Nearest Neighbor	Nearest Neighbor	Nearest Neighbor
uri	http://krig.com	http://krig.com	http://krig.com
description	Single Processor Kriging	Multiprocessor Kriging	None

5.1.3 mergeddataset

A merged data set is a pool of data from which a modelarray is extracted. Is the complete dataset for a study site.

Table Definition:

Column Name	Type	Modifiers	Description
name	varchar	not null	The name of the data set.
studysitename	varchar	not null	The foreign key for a row in the study site table.
uri	varchar	not null	Currently not used.
variablesname	varchar	not null	Variable name that is contained in the data set. Currently there is only one variable name. This table must be changed to support many variable names.
description	varchar	not null	A detailed description of this data set.

Example Record:

Column Name	Example Cerro Grande	Example Rocky Mountain
name	Rocky_Mtn_Data	cerroGrande_data
studysitename	Rocky_Mtn_Np	cerroGrande
uri	http://www.nps.gov/romo/	none
variablesname	total plants	total plants
description	Rocky Mountain Merged Data Set	Cerro Grande Merged Data Set

5.1.4 modelarray

Subset of the mergeddataset used to run against the kriging algorithm. Contains variables from the mergeddataset that may be predictors in the forecasting of scenarios.

Table Definition:

Column Name	Type	Modifiers	Description
name	varchar	not null	The name of the model array.
mergeddataname	varchar	not null	Specifies which data set this model array was generated from. The foreign key for a row in the mergereddataset table.
url	varchar	not null	Currently not used.
predictorvar	varchar	not null	The variable name to use as a predictor. Cur rently we only support one. This table will have to change to support
description	varchar	not null	A detailed description.
version	varchar	not null	Each model array is versioned.

Example Record:

Column Name	Example Cerro Grande	Example Rocky Mountain
name	CG_MA	rmnp1810sub-TC
mergeddataname	cerroGrande_data	Rocky_Mtn_data
url	webapps/isfs/data/studySites/cerroGrande/cerroGrande.ma	webapps/isfs/data/studySites/rmnp/rmnp_modelArray.ma

predictorvar	total plants	total plants
description	Model array for Cerro Grande	Model array for RMNP
version	1.0	1.0

5.1.5 modeloutput

Holds the console output from the model run and descriptors for the model run such as a timestamp and a locator for the image produced.

Table Definition:

Column Name	Type	Modifiers	Description
modelname	varchar	not null	The logical name of the modeloutput. The name is the same as the name of the modelrun. It is a foreign key for a row in the modelrun table.
outputinfo	Text	not null	Text from the modelrun output.
timegenerated	varchar	not null	Timestamp of the modelrun.
imageurl	varchar	not null	Created from the modelrun output image filename.

Example Record:

Modeloutput is a large string that is IDL standard output. This will be improved at some point to handle some type of structured metadata record.

5.1.6 modelrun

Holds parameters for specifying a model run. A row will exist for each model run requested on the system.

Table Definition:

Column Name	Type	Modifiers	Description
name	varchar	not null, unique	A unique logical name for a particular model run.
modelarrayname	varchar	not null	The name of the model array used in this model run. A foreign key to the modelarray table.
krigname	varchar	not null	Specifies the krig routine used in this model run. Foreign key to krigroutine table.
description	text		An optional text description for a particular model run.
hasrun	boolean		True if the model run has been executed and finished.

Example Record:

Column Name	Example
name	1058811086851
modelarrayname	rmnp1810sub-TC
krigname	MP_Fortran
description	<blank>
hasrun	T

5.1.7 studysite

This table is an index of study sites for which we have a data set.

Table Definition:

Column Name	Type	Modifiers	Description
name	varchar	not null	The name of the study site. Currently this is the logical name used by the kriging algorithm.
uri	varchar	not null	Image identification reference tool.
description	text		A detailed description of the site.

Example Record:

Column Name	Example Cerro Grande	Example Rocky Mountain
name	cerroGrande	Rocky_Mtn_Np
uri	None	http://www.nps.gov/romo/
description	Cerro Grande	Rocky Mountain National Park

5.1.8 useraccount

The useraccount table holds user information. Currently the passwords are not encrypted.

Table Definition:

Column Name	Type	Modifiers	Description
accountname	varchar	not null	The username. This should be the user's email address for notification to work.
password	varchar	not null	The plain-text password for the user. We may want to encrypt it in the future.

Example Record:

Column Name	Example
accountname	Jdoe
password	********

5.2 Entity Relationship (ER) Diagram

This figure illustrates the relationship between database tables (entities). In the diagram, rectangles represent entities and diamonds represents the relationship between entities. The diagram should be read from left to right and top to bottom.

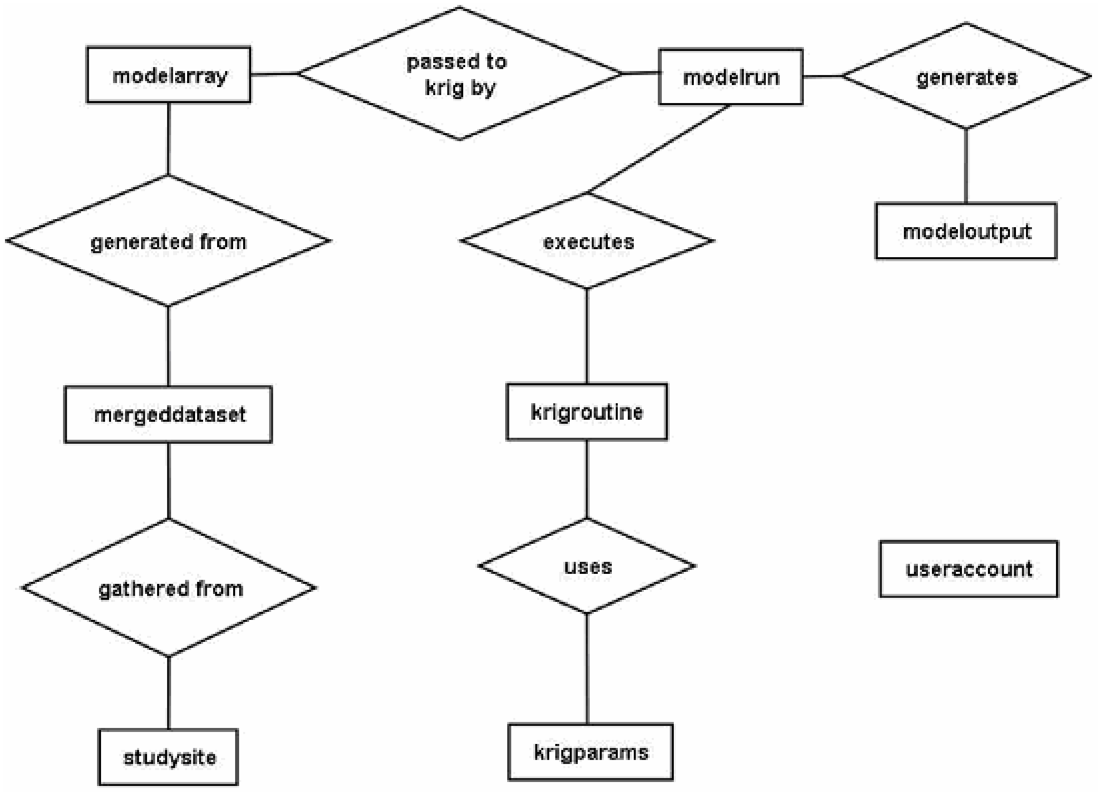


Figure 13 - Entity Relation Diagram

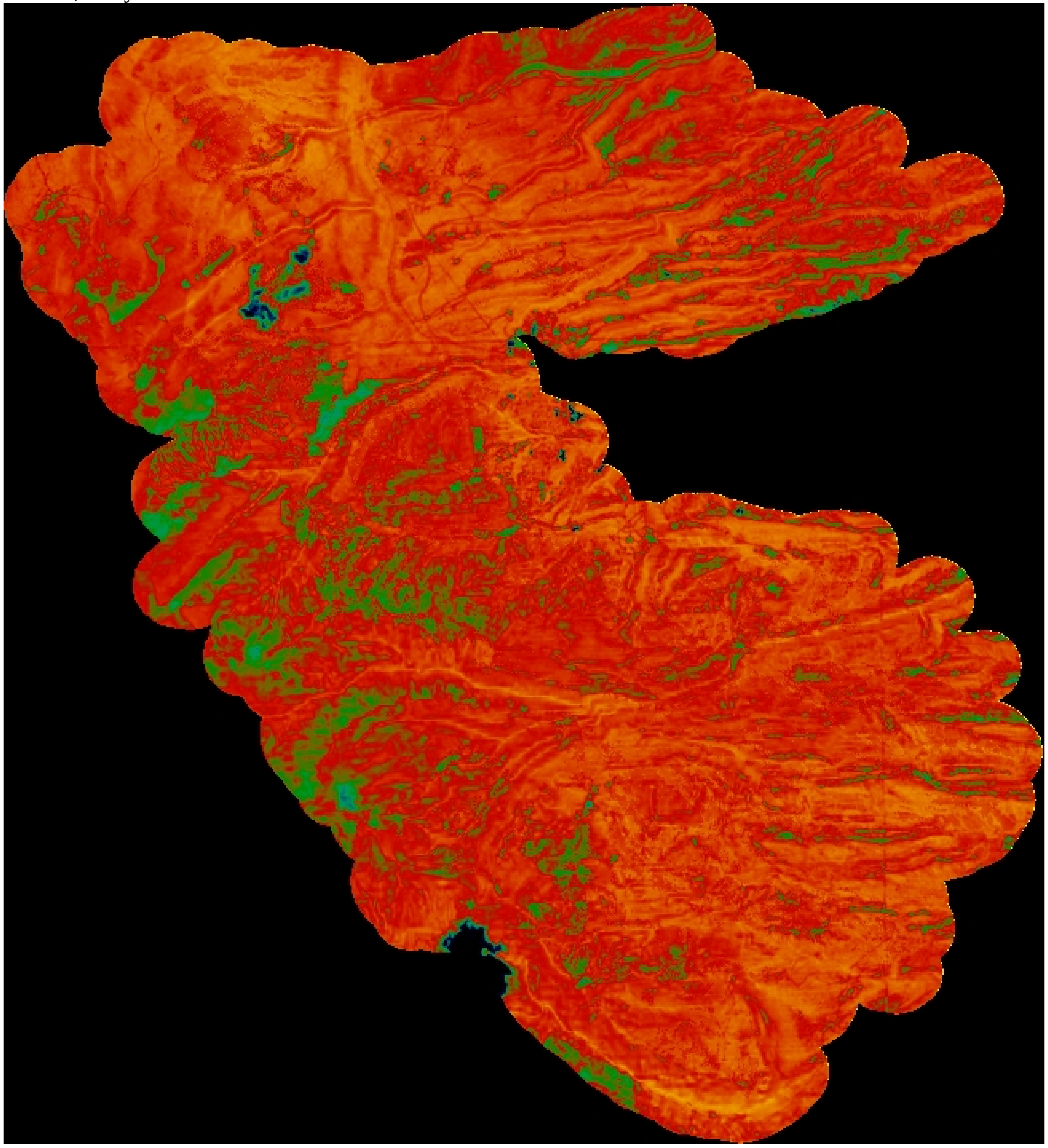
Glossary

BP	Biotic Prediction project
CONOP	Concept of Operations
COTS	Commercial Off The Shelf
CSU	Colorado State University
CT	Computational Technologies project
CVS	Concurrent Version Software
DAAC	Distributed Active Archive Center
DAO	Data Access Object
DEM	Digital Elevation Models
DHTML	Dynamic Hyper-Text Markup Language
DTO	Data Transfer Object
ECHO	Earth Clearing HOuse
ENVI	Environment for Visualizing Images
ESML	Earth Science Markup Language
ESTO	Earth Science Technology Office
ETM+	Enhanced Thematic Mapper Plus
FTP	File Transfer Protocol
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HPC	High Performance Computing
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transport Protocol
IDE	Integrated Development Environment
IDL	Interactive Data Language
ISFS	Invasive Species Forecasting System
J2EE	Java 2 Platform, Enterprise Edition
J2SE	Java 2 Platform, Standard Edition
JDBC	Java Database Connectivity
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JOX	Java Objects in XML
JSP	Java Server Pages
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MODIS	MODerate resolution Imaging Spectroradiometer
MVC	Model View Controller
NASA	National Aeronautic and Space Association
NBII	National Biological Information Infrastructure
NOAA	National Oceanic and Atmospheric Administration
NREL	Natural Resources Ecology Laboratory
OLS	Ordinary Least Squares
RDBMS	Relational Database Management System
SCP	Secure Copy
SDD	Software Design Document
SOAP	Simple Object Access Protocol
SRD	Software Requirements Document
SSH2	Secure Shell
UML	Unified Modeling Language
URL	Uniform Resource Locator
USGS	United States Geological Survey
XML	Extensible Markup Language

Appendix A Input/Output Samples

Map Output

This is a map produced by the modeling application of plant diversity in the Cerro Grande Fire Site in Los Alamos, study site.



Logged Output from Stepwise Regression with Kriging

The output presented below was run on the FRIO cluster on November 24, 2003. A test program "*proto_ISFS*", was utilized in this model run. This program contains the same code that is launched when the model is run w/in the web application. This program provides the following services:

- Ensures required ENVI/IDL libraries & routines are compiled & loaded
- Performs simple parameter validation
- Loads & interprets merged dataset
- Outputs & logs routine messaging
- Calls routines in IDL/ENVI which run the distance matrix, modeling technique, variogram & spatial fitting. The Kriging program is also called, which is written in Fortran77.
- Generates geographical output

```
IDL Version 5.6 (linux x86 m32). (c) 2002, Research Systems, Inc.
Installation number: 10045.
Licensed for use by: NASA/GSFC

% Restored file: ENVI.
% Loaded DLM: TIFF.
% Restored file: ENVI_M01.
% Restored file: ENVI_M02.
% Restored file: ENVI_M03.
% Restored file: ENVI_M04.
% Restored file: ENVI_M05.
% Restored file: ENVI_M06.
% Restored file: ENVI_M07.
% Restored file: ENVI_M08.
% Restored file: ENVI_D01.
% Restored file: ENVI_D02.
% Restored file: ENVI_D03.
% Restored file: ENVI_CW.
% Restored file: ENVI_IDL.
% Restored file: ENVI_IOU.
..run proto_ISFS
..run stepreg
..run varfuncs
proto_ISFS
ENVI> .run proto_ISFS
% Compiled module: PROTO_ISFS.
ENVI> .run stepreg
% Compiled module: STAT_REGR_OUT.
% Compiled module: MY_STEPWISE.
ENVI> .run varfuncs
% Compiled module: EXPF.
% Compiled module: GAUF.
% Compiled module: SPHERF.
ENVI>
```

1. IDL is launched.
2. Required routines & libraries are loaded (i.e. Restored). These are primarily related to the ENVI application, which is written in the IDL programming language.
3. Programs are run, each module is then compiled and thus ready to run w/in ENVI
4. 'stepreg' runs the stepwise regression modeling technique
5. 'varfuncs' generates the variogram

```
ENVI> proto_ISFS
base filename = /home/jpedelty/CT/rmnp/proto/rmnp1810sub-tp1
```

1. *proto_ISFS* ENVI program is launched.
2. parameter file reading. This file contains location & filename of merged dataset, identifies which columns contain the X & Y coordinate values, locates imagery data file associated with each predictor var

```
**************************************************
Read the input file containing the merged field and RS data
**************************************************
number of variables =           13
number of lines of data =         1800
% Compiled module: STRPARSE.
variables:
column        0: variable = tplant
column        1: variable = band1
column        2: variable = band2
column        3: variable = band3
column        4: variable = band4
column        5: variable = band5
column        6: variable = band6
column        7: variable = band7
column        8: variable = elev
column        9: variable = slp
column       10: variable = absasp
column       11: variable = xutm
column       12: variable = yutm
location of xutm =          11
location of yutm =          12
```

22. Merged dataset is read, interpreted & key metrics out logged.
23. Total Plant Species (*tplant*) is the Response variable here. Field measurements at each X, Y coordinate have been provided for this field
24. Variable 1-10 are Predictor variables obtained from remote sensed (RS) data
25. Variable 12 & 12 are the X & Y coordinates in UTM projection

```
**************************************************
Computing distance matrix
**************************************************
 XUTM            DOUBLE    = Array[1800]
YUTM            DOUBLE    = Array[1800]
% Compiled module: W.
max before rescaling =       36405.2
min before rescaling =       2.25375
max after rescaling =       16153.2
min after rescaling =       1.00000

**************************************************
Determine boundaries of study area
**************************************************

map:   xl, xu, yl, yu =    422594.75   458132.74  4450614.29  4481848.75
field: xl, xu, yl, yu =    426481.00   456026.35  4451880.00  4476650.00
krigsize =         1186        1041
```

20. Distance Matrix is generated containing the distance of each point to every other point.
21. A rectangular bounding box around the study area is calculated

```
***************************************************
Prepare for modeling steps
***************************************************

xloc[0] =          11
yloc[0] =          12
nvars =          13
elem =       1       2       3       4       5       6       7       8
       9      10
first line of x =       58.000000       22.000000       20.000000
       70.000000       61.000000       125.00000       20.000000
       2709.0000       2.1343000       153.43500
secondline of x =       55.000000       19.000000       19.000000
       56.000000       47.000000       124.00000       16.000000
       2709.0000       2.1343000       153.43500
Y              DOUBLE    = Array[1800]
1st five elements of y =       7.0000000    8.0000000    9.0000000
       7.0000000       10.000000
       0  band1
       1  band2
       2  band3
       3  band4
       4  band5
       5  band6
       6  band7
       7  elev
       8  slp
       9  absasp
```

16. program structures are created & initialized in preparation for running the model.
17. Initial structure contents logged for debugging purposes.
18. First 2 lines of all the Predictor (x) variables *band1, band2, etc*, are logged
19. First 7 values of the Response (y) variable *tplant*, are logged.

```
***************************************************
Perform stepwise regression
***************************************************
% Compiled module: STDDEV.
% Compiled module: MOMENT.
% Compiled module: REGRESS.
% Compiled module: T_CVF.
% Compiled module: T_PDF.
% Compiled module: IBETA.
Final Statistics
Variables in fit:  band5 elev band7 band2 band1 slp band6
 Var       m          R %       S(m)         Beta       Temp  Tprob
 band5   2.41e-01  12.2   1.73e-02   1.36e+00 13.913  100.0
  elev   4.43e-03   8.4   4.45e-04   2.96e-01  9.962  100.0
 band7  -3.83e-01   7.8   3.96e-02  -1.21e+00  9.671  100.0
 band2  -2.52e-01   7.0   9.11e-02  -2.88e-01  2.765   99.7
 band1   1.56e-01   5.9   6.04e-02   2.65e-01  2.585   99.5
   slp  -5.52e-02   3.5   1.54e-02  -7.90e-02  3.590  100.0
 band6   5.46e-02   2.2   1.93e-02   9.57e-02  2.832   99.8

 Const      -18.937  R  24.2%  F_emp    81.665
% Compiled module: F_PDF.
 F_prob  100.0  J    7 DF 1792  n 1800

InEQ =        4       7       6       1       0       8       5
xvarnames(InEQ) =  band5 elev band7 band2 band1 slp band6
result =       0.24134670
    0.0044325691
     -0.38294379
     -0.25177076
      0.15618168
    -0.055248675
     0.054570118
const =       -18.937047
```

13. Stepwise Regression modeling algorithm is run by the *stepreg* program. Note: the regression may not converge. In this case, an error message will be generated and processing will stop.
14. The set of input predictor variables that are identified as significant are listed in the "Final Statistics" section along with their coefficients & R^2
15. The linear model is presented in the "result" line. In this example the linear formula to predict Total Plant Species is:

tplants = 0.24134670 (band5) +
0.0044325691 elev) +
-0.38294379 (band7) +
-0.25177076 (band2) +
0.15618168 (band1) +
-0.055248675 (slp) +
0.054570118 (band6) +
-18.937047

```
**************************************************
Calculate and plot the variogram of the residuals to the fit
**************************************************
 min, max of residuals =       -10.625395       17.661105
% Compiled module: DOVARIO.

 GAMV Version: 2.000

  data file = ./residuals.dat
  columns for X,Y,Z =  1 2 0
  number of variables =  1
  columns =  3
  trimming limits =  -1.00000002E+21  1.00000002E+21
  output file = ./gamv.out
  number of lags =  20
  lag distance =   1000.
  lag tolerance =   0.
  number of directions =  1
  azm, atol, bandwh =   0.  180.  9999.
  dip, dtol, bandwd =   0.  90.  90.
  flag to standardize sills =  0
  number of variograms =  1
  tail,head,type =  1 1 1

 xltol is too small: resetting to xlag/2
 Variable number  1
   Number   =  1800
   Average  =   5.59815394E-09
   Variance =   25.2956276

 Variogram  1 Semivariogram         :  tail=Residual     head=Residual

 GAMV Version: 2.000 Finished
```

11. Generate a variogram, plotting residuals for each point as a function of the distance to all other points.
12. Number '1800' should equal Total Data Points as indicated in 'number of lines of data' line when the Merged Dataset was read.

```
% Compiled module: READCOL.
% Compiled module: NUMLINES.
% READCOL: Format keyword not supplied - All columns assumed floating point
% Compiled module: GETTOK.
% Compiled module: REPCHR.
% READCOL: Skipping Line 1
% Compiled module: STRNUMBER.
% READCOL: 22 valid lines read
vsize =           22
% Compiled module: MEAN.
initial fit guess =       32.7029      9982.05
% Compiled module: CURVEFIT.
Gaussian fit coefficients =       26.9169      542.588
Exponential fit coefficients =       26.9221      549.740
Spherical fit coefficients =       26.9169      401.378
dokrg =    1
gau
```

10. Determine best method [Gaussian, Exponential or Spherical] to fit the variogram to the model. This will model residuals for all the non-observed points, based on the calculated residuals from the field points. Basically, filling in the gaps.

```
************************************************
Perform the kriging of the residuals
************************************************
 Kriging routine = Reich Fortran
% Compiled module: MAKEKRIG.
percent done =       0
percent done =       5
percent done =      10
percent done =      15
percent done =      20
percent done =      25
percent done =      30
percent done =      35
percent done =      40
percent done =      45
percent done =      50
percent done =      55
percent done =      60
percent done =      65
percent done =      70
percent done =      75
percent done =      80
percent done =      85
percent done =      90
percent done =      95
percent done =     100
Done with kriging
ncols: 1186
nrows: 1041
xllcorner: 422594.750000
yllcorner: 4450614.500000
cellsize: 29.965000
max is 34.078171 and min is -35.102051
ncols: 1186
nrows: 1041
xllcorner: 422594.750000
yllcorner: 4450614.500000
cellsize: 29.965000
max is 7.219118 and min is 0.737369
% Restored file: ENVI_UTL.
```

8. Kriging Fortran routine is called. Progress is reported as a percentage complete when run on one processor. No percent progress is calculated when run against multi-processors.
9. These figures should match from the ‘Boundaries of the Study Area’ section.

```
**************************************************
Apply the model to estimate total plants over the study area
**************************************************
InEQ =        4       7       6       1       0       8       5
number of significant variables =            7
coefficients =       0.24134670
    0.0044325691
     -0.38294379
     -0.25177076
      0.15618168
    -0.055248675
     0.054570118
opening and reading image file band5
opening and reading image file elev
opening and reading image file band7
opening and reading image file band2
opening and reading image file band1
opening and reading image file slp
opening and reading image file band6
Min and max of the derived total plant image =       0.00000      43.4447
       0: fid_name = /home/jpedelty/CT/rmnp/proto/rmnp1810sub-tp1_total_plants
Displaying image /home/jpedelty/CT/rmnp/proto/rmnp1810sub-tp1_total_plants
Warning:
    Name: button848
    Class: XmRowColumn
    XtGrabPointer failed.

       1: fid_name = ./image-data/band6
       2: fid_name = ./image-data/slp
       3: fid_name = ./image-data/band1
       4: fid_name = ./image-data/band2
       5: fid_name = ./image-data/band7
       6: fid_name = ./image-data/elev
       7: fid_name = ./image-data/band5
       8: fid_name =
/home/jpedelty/CT/rmnp/proto/rmnp1810sub-tp1_Reich_krig_error
       9: fid_name =
/home/jpedelty/CT/rmnp/proto/rmnp1810sub-tp1_Reich_krig_resid
      10: fid_name = ./image-data/studyarea
ENVI>
ENVI> Warning:
    Name: menubar
    Class: XmRowColumn
    XtGrabPointer failed.

% Interrupted at: WIDGET_BAND_LIST_UPDATE_TREE
ENVI> exit
```

1. Apply the model over the entire study area as defined by the *rmnp1810sub-tp1_total_plants*
2. Predictor variable order shown in InEQ field should match from Stepwise Regression step.
3. Logged coefficients should match those output in the Stepwise Regression step.
4. The remote data associated with Predictor variables are loaded.
5. Then the image of the study area is loaded.
6. Krigged errors & residuals are loaded & applied to study area
7. Image studyarea is displayed

Merged Dataset Sample

A Merged Dataset concatenates the measured variable with remote sensed image data at each geographic coordinate in the measured field data. This snippet merges the response field [*tplant*] (i.e. field data) with the predictor variables [*band1-7*, *elev slp & absasp*] (i.e. remote sensed, image data) at each measured geographic coordinate, in UTM projection [*xutm* & *yutm*].

tplant	Band1	band2	band3	band4	band5	band6	band7	elev	slp	absasp	xutm	yutm
7	58	22	20	70	61	125	20	2709	2.1343	153.435	426531	4467419
8	55	19	19	56	47	124	16	2709	2.1343	153.435	426522	4467407
9	55	19	19	56	47	124	16	2710	0.337615	135	426510	4467407
7	55	20	18	64	49	124	14	2710	1.06752	63.43495	426481	4467415

tplant	Band1	band2	band3	band4	band5	band6	band7	elev	slp	absasp	xutm	yutm
10	55	20	18	64	49	124	14	2710	0.337615	135	426490	4467427
2	55	19	19	56	47	124	16	2710	0.337615	135	426514	4467427
4	55	19	19	56	47	124	16	2710	0.337615	135	426516	4467418
9	55	19	19	56	47	124	16	2710	0.337615	135	426513	4467415
8	55	20	18	64	49	124	14	2710	0.337615	135	426496	4467415
9	55	19	19	56	47	124	16	2710	0.337615	135	426503	4467420
8	57	23	22	62	69	129	26	2708	2.361991	135	426541	4467053
9	57	23	22	62	69	129	26	2707	2.882181	114.444	426553.7	4467045
6	57	23	22	62	69	129	26	2707	2.901822	80.53768	426554.8	4467033
13	59	24	22	76	69	128	26	2707	2.569503	21.80141	426549.3	4467003
12	59	24	22	76	69	128	26	2708	1.217118	78.69006	426536.6	4467011
6	57	23	22	62	69	129	26	2708	1.217118	78.69006	426534.5	4467035
7	57	23	22	62	69	129	26	2708	1.217118	78.69006	426543.8	4467038
7	57	23	22	62	69	129	26	2708	1.217118	78.69006	426547	4467035
10	59	24	22	76	69	128	26	2708	1.217118	78.69006	426548	4467018
11	59	24	22	76	69	128	26	2708	1.217118	78.69006	426542.9	4467025
3	53	19	18	45	45	122	14	2929	4.393377	77.47119	432528	4463861
3	53	19	18	45	45	122	14	2930	6.457954	173.6598	432514.2	4463855
6	51	17	16	41	33	123	11	2927	4.044691	135	432503.8	4463861
4	53	21	21	60	59	123	18	2926	5.788832	170.5377	432482.7	4463883
4	52	21	21	62	50	122	17	2927	4.044691	135	432496.5	4463889
4	54	23	22	72	56	121	17	2927	4.044691	135	432517.3	4463877
3	54	23	22	72	56	121	17	2927	4.044691	135	432514.3	4463867
3	52	21	21	62	50	122	17	2927	4.044691	135	432510.2	4463866
6	52	21	21	62	50	122	17	2927	4.044691	135	432495.7	4463875
3	52	21	21	62	50	122	17	2927	4.044691	135	432504	4463876

Model Run Params XML

Below is an example of the model parameters file that the application generates for a model run. This file is passed to the compute server to process the model. The XML file includes all input file names and input parameter values along with other pertinent descriptive fields needed by the model routines. This file is expected to grow as the application design matures.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ISFS_RUN>
<doKriging>false</doKriging>
<krigParams>0</krigParams>
<krigRoutine>none</krigRoutine>
<krigRoutineLabel>None</krigRoutineLabel>
<modelArray>CG_MA</modelArray>
<runId>1069868804954</runId>
<servletWrapper />
<studySite>cerroGrande</studySite>
<studySiteLabel>Cerro Grande</studySiteLabel>
<willWait>true</willWait>
</ISFS_RUN>
```

Appendix B Developer Setup/Deployment Instructions

Tools and Technologies Overview

Software included in the ISFS development environment includes the following:

- Ant — ISFS uses Ant as a build tool. ANT can be downloaded from http://ant.apache.org/
- Beowulf Cluster — Parallel computer with a cluster of PCs running connected by its own private LAN.
- Bugzilla — The Bugzilla Defect Tracking System tracks bugs and enhancement requests.
- CVS — Concurrent Version System configuration management software is used for file backup and version control of developers' files including all .html pages, .jsp's, scripts, images, and documentation.
- ENVI — Remote sensing software that includes hyperspectral analysis.
- IDL — (Interactive Data Language) software is used for data analysis, visualization, and crossplatform application development. IDL is the underlying language for ENVI.
- Jakarta Struts — ISFS is using the Jakarta Struts web application framework for design and implementation. Struts handles the J2EE code, JSPs, servlets, and Java code. Details about Struts can be found at http://jakarta.apache.org/struts/.
- Java — J2SE 1.4 is required and is available at the Java website: http://java.sun.com/
- JCraft — SSH2 Terminal Emulator in Pure Java — http://www.jcraft.com/jcterm/
- JOX — "JOX is a set of Java libraries that make it easy to transfer data between XML documents and Java beans." — http://www.wutka.com/jox.html
- Poolman — The PoolMan library and JDBC2.0 Driver and DataSource provide a JMX-based, XML-configurable means of pooling and caching Java objects, as well as extensions for caching SQL queries and results across multiple databases.
- PostgreSQL — The database used for ISFS is a Postgres database and the name of the database is isfs. The schema file is located in the /BP/bin folder and is named isfs.sql. This is an SQL script used to initialize the database.
- TogetherJ — TogtherJ is the project's UML/IDE tool. The TogetherJ project file, ISFS.tpr, is located in the BP folder of the project.
- Tomcat — Tomcat provides the Servlet and JSP engine that ISFS uses to run the Portal. To get Tomcat, the open source software can be downloaded from http://jakarta.apache.org/tomcat/. Currently the project is using Tomcat 4.1.18, but a more current version should work fine. The web.xml file for ISFS is located in the /BP/config folder in case you need to alter that file.

Tools and Environment Setup

WinCVS

Installation and Configuration: The ISFS project uses CVS for source control. You must install and setup a CVS client in order to access CVS. For Win32, MacOSX and Unix/Linux systems CVSGUI is available at the following location http://wincvs.org/.

The ISFS projects CVS repository is available at: http://carbon.sesda.com/cgi-bin/bp/viewcvs.cgi/BP/?sortby=date. The user may use a browser to access the version history of the project files. WinCVS may be obtained at http://wincvs.org

Configuring a CVS Client

The CVS repository is located on the server carbon.sesda.com. To obtain write access to the repository contact David.J.Kendig.1@gsfc.nasa.gov. After an account has been established, set up your client to use CVSROOT=pserver:myaccount@carbon.sesda.com:/export/home/cvs where myaccount is your personal cvs account name. The repository module is named 'BP'. Example:

```
cvs -d pserver:myaccount@carbon.sesda.com:/export/home/cvs login
cvs -d pserver:myaccount@carbon.sesda.com:/export/home/cvs checkout BP
```

Java

J2SE 1.4 is required and is available at the Java website: http://java.sun.com/

Ant

ISFS uses Ant as a build tool. To get ANT you can download from: http://ant.apache.org/. The Ant build file for ISFS is located in the /BP/bin folder. Read this file to see what targets exits. In order to run Ant you must have the Environment Variables "JAVA HOME" set equal to the path to Java 1.4 and "ISFS HOME" set to equal the `path_to_tomcat_installation\webapps\isfs.` The isfsant.bat file is located in the /BP/bin folder and should be edited with your path to the Ant executable, or you can use an environment variable, your choice. The batch file takes the target options and passes it to the executable, for example "> isfsant compile" will compile the ISFS project using Ant.

Tomcat

Tomcat provides the Servlet and JSP engine that ISFS uses to run the Portal. Tomcat may be downloaded from http://jakarta.apache.org/tomcat/. Currently we are using Tomcat 4.1.18, but more current version should work fine. It is also possible to use any other Servlet and JSP engine of your choice but currently we use Tomcat. The web.xml file for ISFS is located in the /BP/config folder in case you need to alter that file.

TogetherJ

The ISFS project has a copy of TogtherJ which is used as the UML/IDE tool. You are not required to use TogetherJ as your chosen tool, however to take advantage of the engineering information to date, you should use TogetherJ. The TogetherJ project file, ISFS.tpr, is located in the BP folder of the project.

Database

The database used for ISFS is a Postgres database that is running on port 5432 on the carbon.sesda.com server. The name of the database is isfs. The schema file is located in the /BP/bin folder and is named isfs.sql. The poolman config file needs editing if you switch databases. This file is described in more detail in the "Configuration files" section below. To login to the database on carbon, login to carbon and issue the following command: `/usr/local/pgsql/bin/psql -U postgres isfs` which will log you in to the "isfs" database as the user "postgres". The Postgres slash commands come in handy for describing info about the database and it's tables.

Struts

ISFS is using the Jakarta Struts web application framework to simplify the design and implementation. Details about Struts can be found at http://jakarta.apache.org/struts/. The Struts config file is located in the /BP/config folder and is described in more detail in the section below. If you have not used Struts before it is recommended that you familiarize yourself with Struts and it's concepts before altering ISFS code. The O'reilly book Programming Jakarta Struts is a good resource to have and the Struts website has plenty of documentation.

Configuring for local development

During the development process, it is possible to use your own local database and Tomcat servers so as not to use the production versions for testing your development. Some changes would need to be made to config files in order to do this.

poolman.xml

ISFS uses poolman to do connection pooling. Poolman's configuration is based on a poolman.xml file. The xml file defines how to connect to each database that ISFS uses in addition to a bunch of parameters that define how the pool should function. The file is located at `\BP\config\poolman.xml.` You need to:

- alter the entry for the <!-- `Standard JDBC Driver info -->` section of the datasource element so that it points to your database.
- Change `<dbname>` to the name of the database on your local db server.

- Change: `<url>jdbc postgresql://carbon.sesda.com:5432/isfs</url>` to: `<url>jdbc:postgresql://machine_name:port/datbase_name</url>`
- Change the user name and password appropriately.
- If you are using a database other than Postgres. ensure you have the appropriate driver in the /BP/lib folder

isfs.props

The isfs.props file holds properties used by the core of ISFS to figure out locations of servers needed during the running of ISFS models. If you are developing locally (i.e. your own local version of Tomcat or other changes) you must change the values of these properties. For example, if you are running a local Tomcat server, then you should change `WEBSERVER_HOME=full_path_to_tomcat_installation` `WEB_HOST=http://localhost:8080`

Description of props:

`WEBSERVER_HOME` – full path to web server home
`WEBSERVER_DATA_FOLDER` – path to folder on web server containing the modelarray data.
`WEBSERVER_IMAGES_FOLDER` – path to folder on web server containing output images.
`SCRIPT_LOCATION` – location on server host containing kriging script
`SCRIPT_NAME` – name of the kriging script
`USER` – user name for the sever
`SERVER_HOST` – kriging server host
`SERVER_IN_FOLDER` —where the input data should be put for the run
`SERVER_OUT_FOLDER` – where the output data will be put after the run `WEB_HOST` —the url to the web application host
`MAIL_FROM` —the from line in emailed modelrun jobs
`MAIL_SERVER` —name of the server that sends modelrun emails

The following config files are independent of sever development location but are config files nonetheless.

web.xml

The web.xml file is used by the Servlet and JSP engine(Tomcat) to determine servlet request mappings. The web.xml file used in ISFS has Struts specific information in it and should only be altered by a developer that is familiar with web.xml files. It is located in the /BP/config folder.

struts-config.xml

The struts-config.xml file is where the struts application framework accesses all of the proper information specific to your struts application. Actions, Exceptions, Forms and Forward definitions are some of the elements in this file. Visit the http://jakarta.apache.org/struts/ site for more information. It is located in the /BP/config folder.

An ISFS example: In the `..\isfs\accounts` package there is a file named LogonAction.java which contains an execute method. This class is an extension of the Struts Action class, the execute method gets called when the request forwards to the struts defined logon action which is defined in the struts-config.xml file like below:

```
<action-mappings> <action
     path="/logon"
     type="gov.nasa.gsfc.isfs.accounts.app.LogonAction"
     name="logonForm"
     scope="request"
     validate="true"
     input="/modelrun/logon.jsp">
     <forward name="success"
     path="/viewsites.do?method=viewSites"/>
</action> </action-mappings>
```

logger.props

The logger.props file contains settings for the logger, such as how fine the logging information needs to be, i.e. show all debug info etc., and the location of the log file.

Running ISFS

Getting source

Use your CVS client (described above) to get the latest, and make sure that you have all of the tools described above.

Compiling

Run the script `\BP\bin\isfsant.bat.`

`isfsant.bat deploy –` will compile the Java code and move all of the ISFS resources to the web server for testing/deployment.

Running

- Make sure that the Database and Mail servers, to which you are connecting, are running.
- Run Tomcat. [Start your Tomcat server by using tomcats start.bat or start.sh]
- Open browser, and connect to ISFS at http://localhost:8088/isfs/modelrun/logon.jsp of course localhost:8088 should be server name and port number depending on your dev setup.

Deployment

Deploying the ISFS application is the same as setting it up for developing except that all of the defaults in the config files are set to work for the deployment machine Carbon.sesda.com, thus the user does not have to make any changes to the config files. If the deployment machine/s changes, then the files will need to be edited like above.

To deploy, follow the instructions above (taking into consideration the above comment) and once you have finished compiling using the “isfsant.bat deploy” command, you should then do the “isfsant.bat war” command to package up the application into a web archive and place it into the \webapps folder on your production machines version of Tomcat. Then restart Tomcat.

Appendix C Model Code Performance Improvement

Explained below is the design, development and results of parallel kriging, which equates to the first phase of the model code performance improvement as defined in the task schedule.

We are working with two “canonical” study sites: the Cerro Grande Fire Site in Los Alamos, NM (CGFS), and the Rocky Mountain National Park, CO (RMNP). These two sites provide contrasting ecological settings and analysis challenges and vary in the types and scales of data used, areas covered, and maturity of the investigation. As described in detail in the Baseline Software Design Document (BP-BSD-1.3), three factors influence the performance of ISFS model code: the size of the output surface area over which kriging occurs (area), the total number of sample points in the data set (pts), and the number of “nearest neighbor” (nn) sample points from the total data set actually used to compute a kriged value for any given point in the output area. When we first began work with colleagues at USGS, a scalar, single-processor run of this model using S-plus took approximately two weeks. The major computational bottleneck in the model is the kriging routine. Solving for the weights in the equations that form the ordinary kriging system uses LU decomposition with backsubstitution to do matrix inversions. The overall computational complexity of ordinary kriging is thus O (n^3) and the time required to compute a result is strongly influenced by the number of sampled data points used to estimate the residual surface across the entire study area.

The overall goal for code improvement is to reduce processing times and increase the amount of data handled by the model. As described in BP-BSD-1.3, increasing the amount of data handled by the model translates into either increasing spatiotemporal resolution or increasing coverage. We first wish to accomplish quantitative improvements in the underlying model that have been agreed upon by the user community as minimal advances needed to improve core capabilities. These goals are driven by the fact that we are building a 32-node cluster in the USGS facility. We refer to these as “Community Improvement Goals.” The ESTO/CT program, however, provides access to greater computational capabilities that can be used to apply this modeling approach to some important and challenging problems that heretofore have been unapproachable. We would therefore like to use CT’s clusters to attain more challenging performance improvement goals at the same time we are accommodating basic needs. We refer to these complementary challenges as “Advanced Improvement Goals.” Table 1 provides a summary of the baseline performance characteristics of the model code as well as the various performance goals anticipated over the course of the project.

Table 1. Current Performance Characteristics and Improvement Goals.

BASELINE SCENARIO		**Sec**	**Min**	**Hrs**	**Days**	
CGFS base 079 pts 79 nn 01x area (S-Plus) (Version 0.0)		-	-	-	-	
CGFS base 079 pts 18 nn 01x area (S-Plus) (Version 0.0) (USGS Actual)		1209600.0	20160.0	336.0	14.0	
CGFS base 079 pts 18 nn 01x area (S-Plus) (Version 0.0) (NASA Estimate)		1608426.0	26807.1	446.8	18.6	
CGFS base 079 pts 18 nn 01x area (FORTRAN) (Version 0.1)		114.5	1.9	0.0	0.0	
CGFS base 079 pts 79 nn 01x area (FORTRAN) (Version 0.1)		3702.6	61.71	1.03	0.04	A
RMNP base 1800 pts 18 nn 01x area (FORTRAN) (Version 0.1)		443.0	7.4	0.1	0.0	B
RMNP base 1800 pts 1180 nn 01x area (FORTRAN) (Version 0.1) (est.)		6812384.0	113539.7	1892.3	78.8	
COMMUNITY IMPROVEMENT (CI) GOALS	**x baseline**	**Sec**	**Min**	**Hrs**	**Days**	
CGFS base 079 pts 79 nn 01x area (Version 1.0 -F)	25.0	148.1	2.47	0.04	0.0	C
CGFS base 790 pts 79 nn 01x area (Version 2.0 -G)	25.0	148.1	2.47	0.04	0.0	D
CGFS base 790 pts 79 nn 10x area (Version 2.0 -G)	2.5	1481.0	24.68	0.41	0.0	E
RMNP base 1800 pts 18 nn 01x area (Version 2.0 -F)	25.0	17.7	3.1	0.0	0.0	C
ADVANCED IMPROVEMENT (AI) GOALS	**x baseline**	**Sec**	**Min**	**Hrs**	**Days**	
CGFS base 079 pts 79 nn 01x area (Version 1.0 -F)	200	18.5	0.31	0.0	0.0	F
RMNP base 1180 pts 1180 nn 01x area (Version 2.0 -G)	1000.0	6812.4	113.5	1.9	0.1	G
RMNP base 11800 pts 1180 nn 01x area (Version 2.0 -G)	1000.0	6812.4	113.5	1.9	0.1	H
RMNP base 11800 pts 1180 nn 100x area (Version 2.0 -G)	10.0	681238.4	11354.0	189.2	7.9	I

A: Proposed CGFS canonical baseline using FORTRAN kriging routine.

B: Proposed RMNP canonical baseline using FORTRAN kriging routine.

C: Milestone F CI Goal -speed up- 75% efficiency, 32 node cluster = 25x speed up

D: Milestone G CI Goal -increased resolution- “sliding window” adaptive selection of 10% of 10x nn from 1x area

E: Milestone G CI Goal -increased coverage- “sliding window” adaptive selection of 10% of 10x nn from 10x area

F: Milestone F AI Goal -speed up- 75% efficiency, 256+ node cluster = 200x speed up

G: Milestone G AI Goal -speed up- 75% efficiency, 1024+ node cluster = 1000x speed up

H: Milestone G AI Goal -increased resolution- “sliding window” adaptive selection of 10% of 100x nn from 1x area

I: Milestone G AI Goal -increased coverage- “sliding window” adaptive selection of 10% of 100x nn from 10x area

Milestone F — First Code Improvement (Parallel Kriging)

Kriging is a spatial interpolator that determines the best linear unbiased estimate of the value at any given pixel in an output surface or image using a weighted sum of the values measured at arbitrary sample locations. It determines the weights and the spatial continuity of the data as measured by the variogram. The scalar kriging algorithm is a double loop over all rows and for each pixel within the row. At each pixel we determine the n nearest neighbor sample points and compute the (n x n) distance matrix containing the Euclidean distance between each sample points, and also compute the (n x 1) distance vector from the pixel to each of the sample points. The Euclidean distances are converted to statistical distances by applying the variogram model to create a covariance matrix and vector. We obtain the kriging weights by multiplying the inverse of the covariance matrix by the covariance vector. The computationally expensive part of kriging is the inversion of the covariance matrix, which is done at each pixel since the nearest neighbor sample points can vary across the kriged surface.

The steps to estimate the value at each pixel are independent of all other pixels. The algorithm is therefore ‘elegantly parallel’ and highly amenable to parallel implementation via domain decomposition; we simply assign to each processor a section of the output kriged surface or image. We chose to decompose the domain along the rows only, i.e. each processor works with full rows of the output surface. This means we can leave unaltered the inner loop over columns. We could decompose into contiguous rows, effectively giving each processor a strip of the output image. Instead, we chose to assign consecutive rows to separate processors. Thus, for a kriging 512 x 512 image using 32 processors, the first processor would be assigned rows 1, 33, 65. . . 449 and 481, while the last processor would calculate rows 32, 64, 96, ... , 480 and 512.

Both domain decompositions are equally load balanced if the number of sample points used in the covariance matrix is always the same at each pixel. This is the case now, but soon we plan to implement an adaptive scheme that will use more points in densely sampled regions and fewer points in sparsely sampled areas. Significant load imbalance would result if we assigned sparsely sampled rows to one processor while assigning densely sampled rows to another processor.

We have implemented parallel kriging in FORTRAN using MPI, the Message Passing Interface. Our code employs a ‘node 0’ controller process and a collection of worker nodes. Prior to execution we copy to each node an input data file containing the dimensions and cell spacing of the output kriged surface, the variogram parameters that describe the spatial structure, and the series of plant diversity measurements (UTM X and Y coordinates and the number of plant species at each location). Each node reads this input data file, computes the kriged estimates for its assigned rows, and then sends each row to node 0. Node 0 only receives the data from the worker nodes, assembles the kriged surface in memory, and writes the final kriged estimates to its local disk.

We overlap the computation with the communication to increase parallel efficiency. When the first row has been calculated, we issue an asynchronous send (MPI ISEND) of this row to node 0. Since this is a non-blocking send, the processor proceeds to calculate the second row. At the end of this row, we issue a wait (MPI WAIT) to insure that the first row has been received by node 0 before proceeding. For the smallest kriged surface we tested (512 x 512) the compute time for each row is over 4 seconds, thus the first row has more than sufficient time to be received and the wait call should also return ‘immediately’ (in reality, the latency time MPI’s implementation of the MPI ISEND and MPI WAIT calls). Meanwhile, node 0 posts a serial set of asynchronous receive calls (MPI IRECV) for each row sent by the worker nodes, followed by a series of waits (MPI WAITS). When the waits are finished, each row of data is copied into the appropriate location within the output kriged array on node 0.

We have evaluated our parallel implementation on two clusters at the NASA Goddard Space Flight Center. Our code was designed for use on the Medusa cluster, on which we met our Community Improvement Goals. Medusa is a 64-node, 128-processor, 1.2 GHz AMD Athlon cluster with 1 GB of memory per node and 2.3 TB of total disk storage. Each node is connected to the others with dual-port Myrinet. Node 0 is frio.gsfc.nasa.gov, a Linux PC with a single 1.2 GHz AMD Athlon processor and 1.5GB memory, which resides on one of our desks and is connected to the Medusa cluster via fiber Gigabit Ethernet. We typically only log into Node 0, and to the user it appears that all calculations are done on Node 0. We have also used Thunderhead to evaluate our Advanced Improvement Goals. Thunderhead is a 512-processor, 2.4 GHz Pentium 4 Xeon cluster with 256 GB of memory and 20 TB of disk storage.

Parallel Kriging Results on Ideally Sized Test Cases

We begin by presenting performance results for four test problems to evaluate the efficiency of the parallel implementation. We held the number of input data points constant at 79 (the size of the field sample data set for the Cerro Grande Fire Site), while the output kriged image size varied from 512^2, 768^2, 1024^2, to 2048^2. These problem sizes are ideally sized in the sense that an equal number of rows are assigned to each processor in all cases. In each case the area kriged was held constant and the pixel size was decreased as the problem size increased.

Table 2 shows the results of this timing study. The processing times shown are elapsed wall-clock time in seconds. As expected, the kriging time increases in direct proportion to the area of the output kriged surface (e.g. the 2048^2 problem ran 16x longer than the 512^2 case). The processing times decreased nearly linearly as the number of processors was increased, as shown in Figure 5. We define the scaling efficiency for N processors as the ratio of the 1-processor to N-processor wall-clock times divided by N. The efficiencies we obtained were excellent, shown in Table 3, ranging from 96–98% when using 32 processors and over 99% when using 16 or fewer processors. The scaling efficiencies dropped slightly for the 64-processor tests, but were still greater than 97% for the 2048^2 problem.

Table 2. Test Case Timing Results (Elapsed Wall Clock Seconds)

Number of Medusa Processors	Size of Kriged Image			
	2048^2	1024^2	768^2	512^2
65	583.9	147.5	84.0	38.4
33	1150.4	289.8	163.8	73.8
17	2285.1	573.89	324.0	144.7
9	4558.4	1142.0	642.8	287.2
5	9083.9	2277.4	1281.3	571.5
3	18190.4	4556.3	2562.0	1140.9
2	36252.7	9079.6	5107.0	2269.1

Figure 5. Scaling Curves for Test Cases on Medusa

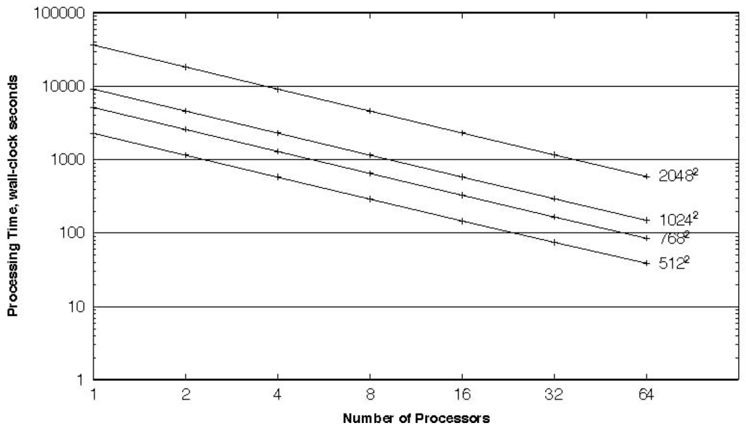


Table 3. Scaling Efficiencies for Test Cases

Number of Medusa Processors	Size of Kriged Image			
	2048^2	1024^2	768^2	512^2
65	97.0%	96.2%	95.0%	92.3%
33	98.5%	97.9%	97.4%	96.0%
17	99.2%	98.9%	98.5%	98.0%
9	99.4%	99.4%	99.3%	98.8%
5	99.8%	99.7%	99.6%	99.3%
3	99.6%	99.6%	99.7%	99.4%
2	100.0%	100.0%	100.0%	100.0%

Evaluation of Community and Advanced Improvement Goals

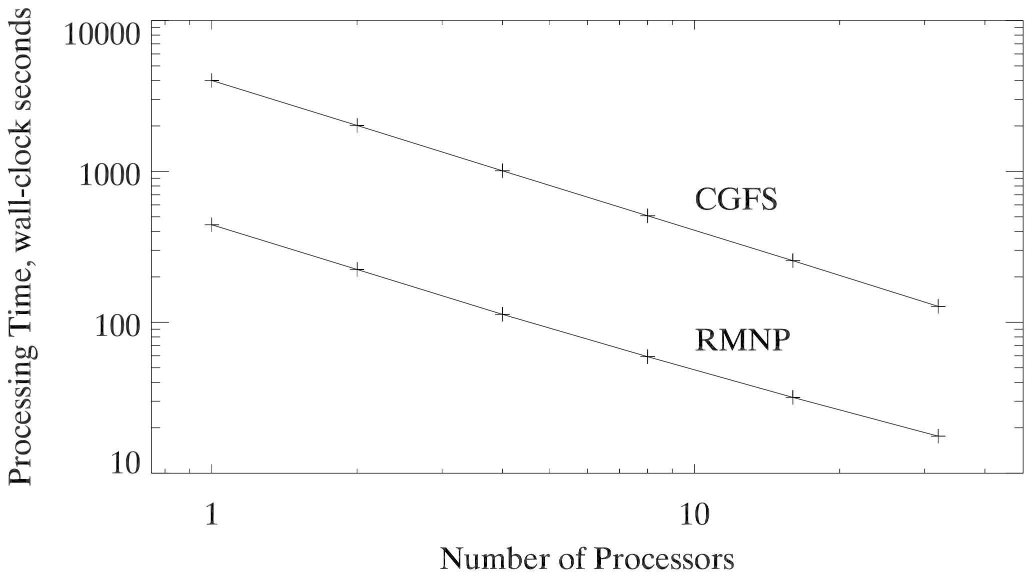
We now evaluate the performance of the parallel kriging on the baseline scenarios defined in Table 1. These differ from the ideally sized test cases evaluated above because a different number of rows are assigned to each processor. As such they represent real operational scenarios that estimate an arbitrary area at a given resolution and use all available processors. Seldom in such scenarios will the number of rows or processors be a power of two, and there will be 'left over' rows that lead to load imbalance and reduced scaling efficiencies.

Table 4. Baseline Scenario Timing and Scaling Results

Number of Medusa Processors	**Baseline Scenario**			
	CGFS (715 rows, 652 cols)		**RMNP (1041 rows, 1186 cols)**	
33	127.5	90.1%	17.6	76.3%
17	255.9	87.1%	31.8	81.9%
9	508.2	82.9%	59.2	83.1%
5	1009.1	75.1%	113.0	66.0%
3	2016.0	62.7%	223.9	98.9%
2	4000.4	52.8%	442.8	50.0%

We show in Table 4 the performance results for both the CGFS and RMNP baseline scenarios, which we plot in Figure 5. We improved the run time for the CGFS test to 2 minutes and 7 seconds, which compares favorably to our Milestone F Community Improvement goal of 2 minutes and 28 seconds. We improved the RMNP test to 17.6 seconds, which is 25.2 times faster than the baseline run time of 7 minutes and 23 seconds. This case can now be run interactively. We can clearly see, however, that the scaling efficiency drops as the run time is reduced. This is expected and due to the parallel overhead and the load imbalance.

Figure 5. Scaling Curves for Test Cases on Medusa



Progress toward Milestones

These results indicate that we have achieved our Milestone F Community Improvement goal of a 25x speed up on a 32-processor cluster with greater than 75% efficiency. The results also document the general scaling behavior of the kriging algorithm and point to the limits in scalability one might expect as more nodes are allocated to the canonical data sets. The project thus completed Milestone F (First Code Improvement) according to plan.